NIST Documentation

Release dev

Marco De Donno

Jun 30, 2021

Contents

1 Table of contents **1**
- 1.1 Installation . . . 1
- 1.2 First steps and examples . . . 2
- 1.3 NIST Traditional module . . . 5
- 1.4 NIST Fingerprint module . . . 6
- 1.5 Plugins manual . . . 44
- 1.6 Changelog . . . 50
- 1.7 Contact . . . 62

References **62**

Python Module Index **63**

Index **64**

Welcome to the documentation of the python NIST library. The aim of the python library is to open, modify and write Biometric Information files based on the standard proposed by the NIST (for short: NIST format) [NIST2013].

The documentation is avaiable on readthedocs (http://nist.readthedocs.io/en/develop/).

This python library contains a main module to read and write NIST files (named 'NIST.traditional'), and a second module to work with fingerprint related NIST files (named 'NIST.fingerprint'). Some complementary modules are provided for some particular cases.

The developpement and maintainance have been made by Marco De Donno, School of Criminal Justice, Faculty of Law, Criminal Justice and Public Administration, University of Lausanne, Switzerland. This library is not related to the National Institute of Standards and Technology.

1 Table of contents

1.1 Installation

The installation process is done as follows.

1. Create a folder on your system to store all the mendatory libraries
2. Clone all mendatory libraries:

```
cd libraries
git clone https://github.com/mdedonno1337/NIST.git
git clone https://github.com/mdedonno1337/MDmisc.git
git clone https://github.com/mdedonno1337/PMlib.git
git clone https://github.com/mdedonno1337/WSQ.git
```

The structure of the files should be as follows:

```
.
├── library
│   ├── MDmisc
│   │   ├── doctester.py
│   │   ├── __init__.py
│   │   ├── LICENSE
│   │   ├── MDmisc
│   │   └── requirements.txt
│   ├── NIST
│   │   ├── CHANGELOG.txt
│   │   ├── doc
│   │   ├── doctester2.py
│   │   ├── doctester.py
│   │   ├── __init__.py
│   │   ├── LICENSE
│   │   ├── NIST
│   │   ├── readme.rst
│   │   ├── requirements.txt
│   │   ├── TODO.txt
│   │   └── update.py
│   ├── PMlib
│   │   ├── LICENSE
│   │   ├── PMlib
│   │   └── requirements.txt
│   └── WSQ
│       ├── __init__.py
│       ├── LICENSE
│       ├── requirements.txt
│       └── WSQ
.
.
.
```

3. Create a `mdedonno.pth` in the site-packages python directory pointing to the libraries folders:

```
$ cat /usr/local/lib/python2.7/dist-packages/mdedonno.pth
/library/NIST
/library/WSQ
/library/PMlib
/library/MDmisc
```

4. Check the installation by running python and typing `import NIST`

1.2 First steps and examples

Traditional NIST objects

Fingerprint NIST objects

The main purpose of this module is to implement some functions needed to work with fingerprint NIST files. The main class is named **'NISTf'**, for NIST-fingerprints.

Examples

Imports:

```
>>> from NIST import NISTf
>>> from NIST.fingerprint.functions import AnnotationList
```

Construction of list of minutiae:

```
>>> lst = [
    [  1, 7.85, 7.05, 290, 0, 'A' ],
    [  2, 13.80, 15.30, 155, 0, 'A' ],
    [  3, 11.46, 22.32, 224, 0, 'B' ],
    [  4, 22.61, 25.17, 194, 0, 'A' ],
    [  5, 6.97, 8.48, 153, 0, 'B' ],
    [  6, 12.58, 19.88, 346, 0, 'A' ],
    [  7, 19.69, 19.80, 111, 0, 'C' ],
    [  8, 12.31, 3.87, 147, 0, 'A' ],
    [  9, 13.88, 14.29, 330, 0, 'D' ],
    [ 10, 15.47, 22.49, 271, 0, 'D' ]
]
>>> minutiae = AnnotationList()
>>> minutiae.from_list( lst, "ixytqd" )
```

Latent fingermark

Construction of the NIST object:

```
>>> mark = NISTf()
>>> mark.add_Type01()
>>> mark.add_Type02()
```

Add the minutiae to the NIST object:

```
>>> mark.add_Type09( 1 )
>>> mark.set_minutiae( minutiae, 1 )
```

Set the core of the latent fingermark:

```
>>> mark.set_cores( [ [ 12.5, 18.7 ] ], 1 )
```

Add a empty image to the NIST object:

```
>>> mark.add_Type13( ( 500, 500 ), 500, 1 )
```

The same result can be obtained by using the following code:

```
>>> params = {
    'minutiae': minutiae,
    'cores': [ [ 12.5, 18.7 ] ]
}
>>> mark = NISTf()
>>> mark.init_latent( **params )
```

The resulting latent fingermark NIST object should be something like:

```
>>> print mark
Information about the NIST object:
    Records: Type-01, Type-02, Type-09, Type-13
    Class:   NISTf
```

(continues on next page)

(continued from previous page)

```
<BLANKLINE>
NIST Type-01
    01.001 LEN: 00000145
    01.002 VER: 0501
    01.003 CNT: 1<US>3<RS>2<US>0<RS>9<US>0<RS>13<US>0
    01.004 TOT: USA
    01.005 DAT: 20161217
    01.006 PRY: 1
    01.007 DAI: FILE
    01.008 ORI: UNIL
    01.009 TCN: 1481995137
    01.011 NSR: 00.00
    01.012 NTR: 00.00
NIST Type-02 (IDC 0)
    02.001 LEN: 00000062
    02.002 IDC: 0
    02.003    : 0300
    02.004    : 20161217
    02.054    : 0300<US><US>
NIST Type-09 (IDC 0)
    09.001 LEN: 00000266
    09.002 IDC: 0
    09.003 IMP: 4
    09.004 FMT: S
    09.007    : U
    09.008    : 12501870
    09.010    : 10
    09.011    : 0
    09.012    : 1<US>07850705290<US>0<US>A<RS>2<US>13801530155<US>0<US>A<RS>3<US>
↪11462232224<US>0<US>B<RS>4<US>22612517194<US>0<US>A<RS>5<US>06970848153<US>0<US>B
↪<RS>6<US>12581988346<US>0<US>A<RS>7<US>19691980111<US>0<US>C<RS>8<US>12310387147
↪<US>0<US>A<RS>9<US>13881429330<US>0<US>D<RS>10<US>15472249271<US>0<US>D
NIST Type-13 (IDC 0)
    13.001 LEN: 00250150
    13.002 IDC: 0
    13.003 IMP: 4
    13.004 SRC: UNIL
    13.005 LCD: 20161217
    13.006 HLL: 500
    13.007 VLL: 500
    13.008 SLC: 1
    13.009 THPS: 500
    13.010 TVPS: 500
    13.011 CGA: 0
    13.012 BPX: 8
    13.013 FGP: 0
    13.999 DATA: FFFFFFFF ... FFFFFFFF (250000 bytes)
```

Note: All dynamic fields (such as the creation date) are calculated by the *NISTf* class, and will (obviously) be different in your case.

Note: All binary fields, such as the field 13.999, are converted to HEX value, and cropped to show the first and last 4 bytes.

Warning: This representation is not usable as input to construct a new NIST file.

1.3 NIST Traditional module

This section contains all the docstring related to the NIST traditional format. This documentation is automaticaly generated, and used as unittest.

NIST module

class NIST.traditional.**NIST**(*init=None*, **args*, ***kwargs*)

Bases: NIST.core.NIST

__module__ = 'NIST.traditional'

clean()

Function to clean all unused fields in the self.data variable. This function should check the content of the NIST file only for fields described in the NIST standard. For all particular implementations and implementation specific fields, overload this function in a new class.

Check done in this function:

- Delete all empty records, IDC and fields
- Recalculate the content of the field 1.003
- Check the IDC field for every ntype (fields x.002)
- Reset all lengths (fields x.001)

dumpbin()

Return a binary dump of the NIST object. Writable in a file ("wb" mode).

Returns Binary representation of the NIST object.

Return type str

hash()

load(*data*)

Load from the data passed in parameter, and populate all internal dictionaries. This function is the main function doing the decoding of the NIST file.

Parameters **data** (*str*) – Raw data read from file.

load_auto(*p*)

Function to detect and load automatically the 'p' value passed in parameter. The argument 'p' can be a string (URI to the file to load) or a NIST object (a copy will be done in the current object).

Parameters **p** (NIST (page 5) *or str*) – Input data to parse to NIST object.

reset_alpha_length(*ntype*, *idc=0*)

Recalculate the LEN field of the ntype passed in parameter. Only for ASCII ntype.

Parameters

- **ntype** (*int*) – ntype to reset.
- **idc** (*int*) – IDC value

reset_binary_length(*ntype*, *idc=0*)

Recalculate the LEN field of the ntype passed in parameter. Only for binary ntype.

Parameters

- **ntype** (*int*) – ntype to reset.
- **idc** (*int*) – IDC value.

write(*outfile*)

Write the NIST object to a specific file.

Parameters **outfile** (*str*) – URI of the file to write to.

Misc functions

1.4 NIST Fingerprint module

This section contains all the docstring related to the NIST traditional format. This documentation is automaticaly generated, and used as unittest.

NISTf module

class NIST.fingerprint.**NISTf**(**args*, ***kwargs*)
Bases: *NIST.traditional.NIST* (page 5)

Overload of the *NIST.traditional.NIST* (page 5) class. This class overload the main class to implement fingerprint oriented functions.

Variables

- **imgdir** (*str*) – Directory to strore the images (minutiae annotations).
- **minutiaeformat** (*str*) – Default minutia format.

__init__(**args*, ***kwargs*)
Constructor function. Call the constructor of the *NIST.traditional.NIST()* (page 5) module, and try to initiate a new latent or print object (see *NIST.fingerprint.NISTf.init_new()* (page 22) for more details).

__module__ = 'NIST.fingerprint'

add_Type04(*idc=1*, ***options*)
Add the Type-04 record to the NIST object.

Parameters **idc** (*int*) – IDC value.

add_Type09(*minutiae=None*, *idc=0*, ***options*)
Add the Type-09 record to the NIST object, and set the Date.

Parameters

- **minutiae** (AnnotationList (page 34)) – AnnotationList with the minutiae data.
- **idc** (*int*) – IDC value.

add_Type13(*size=(500*, *500)*, *res=500*, *idc=0*, ***options*)
Add an empty Type-13 record to the NIST object, and set the resolution (in DPI). Set by default the image to a white image.

Parameters

- **size** (*tuple*) – Size of the latent fingermark image to set.
- **res** (*int*) – Resolution of the image, in dot-per-inch.
- **idc** (*int*) – IDC value.

add_Type14(*size=(500*, *500)*, *res=500*, *idc=1*, ***options*)
Add the Type-14 record to the NIST object.

Parameters

- **size** (*tuple of int*) – Size of the fingerprint image to set.
- **res** (*int*) – Resolution of the image, in dot-per-inch.
- **idc** (*int*) – IDC value.

add_Type15(*idc=1*, ***options*)
Add the Type-15 record to the NIST object.

Parameters **idc** (*int*) – IDC value.

annotate (*image*, *data*, *type=None*, *res=None*, *idc=-1*, ***options*)

Function to annotate the image with the data passed in argument.

Parameters

- **image** (*PIL.Image*) – Non annotated image
- **data** (AnnotationList (page 34)) – Data used to annotate the image
- **type** (*str*) – Type of annotation (minutiae or center).
- **res** (*int*) – Resolution in DPI.

Internal use; not interessting to use directly.

Usage:

```
>>> img_input = sample_type_9_10_14.get_print( 'PIL', 1 )
>>> data = sample_type_9_10_14.get_minutiae( 1 )
>>> data_type = "minutiae"
```

```
>>> img = sample_type_9_10_14.annotate( img_input, data, data_type )
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=RGB size=800x768 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'a816e558611786592fa5bdcb67e018c5'
```

changeResolution (*res*, *idc=-1*)

Change the resolution of the latent fingermark. The minutiae are not affected because they are stored in mm, not px.

Parameters

- **res** (*int*) – Output resolution in DPI
- **idc** (*int*) – IDC value.

Raises **notImplemented** – if the ntype is not 4, 13, or 14.

Usage:

```
>>> sample_type_13.changeResolution( 500 )
```

checkMinutiae (*idc=-1*)

Check if all minutiae are on the image. If a minutiae is outside the image, it will be removed.

Parameters **idc** (*int*) – IDC value.

Returns List of minutiae after clean-up

Return type

AnnotationList

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.
↪80, 15.30, 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [␣
↪4, 22.61, 25.17, 194, 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ],
↪ [ 6, 52.58, 19.88, 346, 0, 'A' ], [ 7, 59.69, 19.80, 111, 0,
↪'C' ], [ 8, 52.31, 3.87, 147, 0, 'A' ], [ 9, 53.88, 14.29, 330,
↪ 0, 'D' ], [ 10, 55.47, 22.49, 271, 0, 'D' ]], format = "ixytqd
↪", type = 'Minutia' )
>>> # Note that the minutiae 5-10 are outisde of the image
>>> sample_type_4_tpcard.add_Type09( idc = 1 )
```

(continues on next page)

(continued from previous page)

```
>>> sample_type_4_tpcard.add_Type09( idc = 2 )
>>> sample_type_4_tpcard.add_Type09( idc = 3 )
>>> sample_type_4_tpcard.add_Type09( idc = 4 )
>>> sample_type_4_tpcard.set_minutiae( minutiae, 1 )
10
>>> sample_type_4_tpcard.set_minutiae( minutiae, 2 )
10
>>> sample_type_4_tpcard.checkMinutiae( 1 ) # doctest:␣
↪+NORMALIZE_WHITESPACE
[
    Minutia( i='1', x='7.85', y='7.05', t='290', q='0', d='A' ),
    Minutia( i='2', x='13.8', y='15.3', t='155', q='0', d='A' ),
    Minutia( i='3', x='11.46', y='22.32', t='224', q='0', d='B'␣
↪),
    Minutia( i='4', x='22.61', y='25.17', t='194', q='0', d='A'␣
↪),
    Minutia( i='5', x='6.97', y='8.48', t='153', q='0', d='B' )
]
```

If not idc is passesd in the function call as argument, all the idc are checked and returned as a list:

```
>>> sample_type_4_tpcard.checkMinutiae() # doctest: +NORMALIZE_WHITESPACE
[[
    Minutia( i='1', x='7.85', y='7.05', t='290', q='0', d='A' ),
    Minutia( i='2', x='13.8', y='15.3', t='155', q='0', d='A' ),
    Minutia( i='3', x='11.46', y='22.32', t='224', q='0', d='B' ),
    Minutia( i='4', x='22.61', y='25.17', t='194', q='0', d='A' ),
    Minutia( i='5', x='6.97', y='8.48', t='153', q='0', d='B' )
], [
    Minutia( i='1', x='7.85', y='7.05', t='290', q='0', d='A' ),
    Minutia( i='2', x='13.8', y='15.3', t='155', q='0', d='A' ),
    Minutia( i='3', x='11.46', y='22.32', t='224', q='0', d='B' ),
    Minutia( i='4', x='22.61', y='25.17', t='194', q='0', d='A' ),
    Minutia( i='5', x='6.97', y='8.48', t='153', q='0', d='B' )
], [
<BLANKLINE>
], [
<BLANKLINE>
]]
```

clean()

Function to clean all unused fields in the self.data variable. This function try to clean the minutiae stored in the current NIST object, and call the *NIST.traditional.NIST.clean()* (page 5) function.

Usage:

```
>>> sample_type_9_10_14.clean()
```

crop(*size*, *center=None*, *ntype=None*, *idc=-1*, ***options*)

Crop the latent or the print image.

Parameters

- **size** (*tuple*) – Size of the output image.
- **center** (*tuple*) – Coordinate of the center of the image, in mm.
- **ntype** (*int*) – ntype to crop (4, 13 or 14).
- **idc** (*int*) – IDC value.

Raises **notImplemented** – if the crop_latent() or crop_print() function raise an notImplemented Exception

Usage:

```
>>> sample_type_13.crop( ( 500, 500 ), ( 12.7, 12.7 ), 13 )
>>> sample_type_4_tpcard.crop( ( 500, 500 ), ( 12.7, 12.7 ), 4, 1 )
>>> sample_type_9_10_14.crop( ( 500, 500 ), ( 12.7, 12.7 ), 14, 1 )
```

crop_auto (**args*, ***kwargs*)

crop_latent (*size*, *center=None*, *idc=-1*, ***options*)

Crop the latent image.

Parameters

- **size** (*tuple*) – Size of the output image.
- **center** (*tuple*) – Coordinate of the center of the image, in mm.
- **idc** (*int*) – IDC value.

Raises **notImplemented** – if the NIST object does not contain Type13 data

Usage:

```
>>> sample_type_13.crop_latent( ( 500, 500 ), ( 12.7, 12.7 ) )
>>> img = sample_type_13.get_latent()
```

```
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=L size=500x500 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'9a23b71585867d11d6a5929a9e004072'
```

crop_print (*size*, *center=None*, *idc=-1*, ***options*)

Crop the print image.

Parameters

- **size** (*tuple*) – Size of the output image.
- **center** (*tuple*) – Coordinate of the center of the image, in mm.
- **idc** (*int*) – IDC value.

Raises **notImplemented** – if the NIST object does not contain Type04 or Type14 data

Usage:

```
>>> sample_type_4_tpcard.crop_print( ( 500, 500 ), ( 12.7, 12.7 ), 1 )
```

export_auto (*f*, *idc=-1*)

export_auto_annotated (*f*, *idc=-1*)

export_latent (*f*, *idc=-1*)

Export the latent fingermark image to a file on disk.

Parameters

- **f** (*str*) – Output file
- **idc** (*int*) – IDC value.

Returns File correctly written on disk

Return type boolean

export_latent_annotated (*f*, *idc=-1*)

Export the latent fingermark annotated to file.

Parameters

- **f** (*str*) – Output file
- **idc** (*int*) – IDC value.

Returns File correctly written on disk

Return type boolean

export_latent_diptych (*f*, *idc=-1*)
Export the latent diptych to file.

Parameters **f** (*str*) – Output file

export_print (*f*, *idc=-1*)
Export the print image to the file 'f' passed in parameter.

Parameters

- **f** (*str*) – Output file.
- **idc** (*int*) – IDC value.

Returns File written on disk.

Return type boolean

Raises **notImplemented** – if the NIST object does not contain Type04 or Type14

Usage:

```
>>> sample_type_4_tpcard.export_print( "/tmp/print.jpeg", 1 )
True
```

export_print_annotated (*f*, *idc=-1*)
Export the annotated print to the file 'f'.

Parameters

- **f** (*str*) – Output file.
- **idc** (*int*) – IDC value.

Returns File written on disk.

Return type boolean

Raises **notImplemented** – if the NIST object does not contain Type04 or Type14

Usage:

```
>>> sample_type_4_tpcard.export_print_annotated( "/tmp/print_annotated.jpeg
↪", 1 )
True
```

export_print_diptych (*f*, *idc=-1*)
Function to export the reference diptych

filter_minutiae (*idc=-1*, *invert=False*, *inplace=False*, **args*, ***kwargs*)
Filter the AnnotationList of minutiae according to the parameters passed as kwarg.

Parameters

- **idc** (*int*) – IDC value.
- **invert** (*boolean*) – Invert (or not) the criteria of filtering
- **inplace** (*boolean*) – Make the changes in-place
- **args** – Positional arguments
- **kwargs** – Keyword arguments

Usage:

```
>>> sample_type_9_10_14.filter_minutiae( d = "AB" ) # doctest: +NORMALIZE_
↪WHITESPACE
[
<BLANKLINE>
]
>>> sample_type_9_10_14.filter_minutiae( d = "CD" ) # doctest: +NORMALIZE_
↪WHITESPACE +ELLIPSIS
[
    Minutia( i='1', x='21.95', y='20.3', t='101', q='00', d='D' ),
    Minutia( i='2', x='21.49', y='20.3', t='96', q='00', d='D' ),
    Minutia( i='3', x='18.59', y='24.0', t='96', q='00', d='D' ),
    Minutia( i='4', x='20.83', y='23.55', t='98', q='00', d='D' ),
    ...
    Minutia( i='45', x='21.44', y='17.81', t='77', q='00', d='D' ),
    Minutia( i='46', x='20.22', y='21.41', t='85', q='00', d='D' ),
    Minutia( i='47', x='21.13', y='21.06', t='89', q='00', d='D' ),
    Minutia( i='48', x='19.71', y='22.23', t='85', q='00', d='D' )
]
```

To get the list filtered by designation, removing Type undetermined (D):

```
>>> sample_type_9_10_14.filter_minutiae( d = "CD", invert = True ) #␣
↪doctest: +NORMALIZE_WHITESPACE
[
<BLANKLINE>
]
>>> sample_type_9_10_14.filter_minutiae( d = "AB", invert = True ) #␣
↪doctest: +NORMALIZE_WHITESPACE +ELLIPSIS
[
    Minutia( i='1', x='21.95', y='20.3', t='101', q='00', d='D' ),
    Minutia( i='2', x='21.49', y='20.3', t='96', q='00', d='D' ),
    Minutia( i='3', x='18.59', y='24.0', t='96', q='00', d='D' ),
    Minutia( i='4', x='20.83', y='23.55', t='98', q='00', d='D' ),
    ...
    Minutia( i='45', x='21.44', y='17.81', t='77', q='00', d='D' ),
    Minutia( i='46', x='20.22', y='21.41', t='85', q='00', d='D' ),
    Minutia( i='47', x='21.13', y='21.06', t='89', q='00', d='D' ),
    Minutia( i='48', x='19.71', y='22.23', t='85', q='00', d='D' )
]
```

To get only the Minutiae id 1 and 5:

```
>>> sample_type_9_10_14.filter_minutiae( i = [ "1", "3" ] ) # doctest:␣
↪+NORMALIZE_WHITESPACE
[
    Minutia( i='1', x='21.95', y='20.3', t='101', q='00', d='D' ),
    Minutia( i='3', x='18.59', y='24.0', t='96', q='00', d='D' )
]
```

get_compression (*idc=-1*)

Get the compression used for a particular image.

Parameters **idc** (*int*) – IDC value.

Returns Compression method

Return type str

Usage:

```
>>> sample_type_4_tpcard.get_compression( 1 )
'WSQ'
```

(continues on next page)

(continued from previous page)

```
>>> sample_type_17_iris.get_compression( 1 )
Traceback (most recent call last):
...
notImplemented
```

get_cores (*idc=-1*)

Process and return the coordinate of the cores.

Parameters **idc** (*int*) – IDC value.

Returns List of cores

Return type *AnnotationList* (page 34)

Usage:

```
>>> sample_type_9_10_14.get_cores() # doctest: +NORMALIZE_WHITESPACE
[
    Core( x='16.66', y='22.78' )
]
```

The function returns 'None' if no cores are stored in the NIST object.

```
>>> sample_type_4_tpcard.get_cores() == None
True
```

get_delta (*idc=-1*)

Process and return the coordinate of the deltas.

Parameters **idc** (*int*) – IDC value.

Returns List of deltas

Return type *AnnotationList* (page 34)

get_diptych (*idc=-1*)

Get the automatic diptych (fingermark or fingerprint).

Parameters **idc** (*int*) – IDC value

Raises **notImplemented** – if the NIST object does not contain Type04, Type13 or Type14 data.

Usage:

```
>>> img = sample_type_13.get_diptych()
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=RGB size=1600x768 at ...>
```

See also:

get_latent_diptych() (page 15) *get_print_diptych()* (page 20)

get_fpc_list ()

Returns a list of unique fpc present in the NIST file, without the idc information. This function will scan all the idc for Type-04, Type-13, Type-14 and Type-15; if no fingerprint data is present or if the NIST transaction file does not contain any fingerprint data, an empty list will be returned. If a Type-04 records has multiple possible FPC, all the values are returned and flattened

```
>>> sample_type_4_tpcard.get_fpc_list()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

```
>>> sample_type_17_iris.get_fpc_list()
[]
```

get_height (*idc=-1*)

Return the height of the image.

Parameters **idc** (*int*) – IDC value.

Returns Height

Return type int

Usage:

```
>>> sample_type_9_10_14.get_height()
768
```

get_idc_for_fpc (*ntype*, *fpc*)

The the first matching IDC for a particular FPC, idcNotFound if the FPC is not available for that ntype.

Parameters

- **ntype** (*int*) – ntype to search ni
- **fpc** (*int*) – FPC to search

Returns first IDC for that FPC

Return type int

Raises

- **idcNotFound** – if the FPC is not in present in that ntype
- **notImplemented** – if the requested ntype is not 4, 13, 14 or 15

```
>>> sample_type_4_tpcard.get_idc_for_fpc( 4, 1 )
1
>>> sample_type_4_tpcard.get_idc_for_fpc( 4, 14 )
11
```

```
>>> sample_type_4_tpcard.get_idc_for_fpc( 4, 25 )
Traceback (most recent call last):
    ...
idcNotFound
```

```
>>> sample_type_4_tpcard.get_idc_for_fpc( 14, 1 )
Traceback (most recent call last):
    ...
idcNotFound
```

get_image (**args*, ***kwargs*)

Get the appropriate image (latent fingermark, fingerprint or palmair image).

Parameters

- **format** (*str*) – Format of the returened image.
- **idc** (*int*) – IDC value.
- **fpc** (*int*) – FPC (Finger Position Code) value.

Returns Fingermark of fingerprint Image

Return type PIL.Image or str

Raises **notImplemented** – if no Type13, Type04 or Type14 data

Usage:

```
>>> img = sample_type_13.get_image()
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=L size=800x768 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'bd5641d4006e395c4e5d7459d2485891'
```

If no image is available, the function will raise an notImplemented Exception.

```
>>> sample_type_13.delete_ntype( 13 )
>>> sample_type_13.get_image()
Traceback (most recent call last):
...
notImplemented
```

get_image_annotated(*idc*)

get_latent(*format='PIL'*, *idc=-1*)

Return the image in the format passed in parameter (RAW or PIL).

Parameters

- **format** (*str*) – Format of the returned image
- **idc** (*int*) – IDC value.

Returns Latent image

Return type PIL.Image

Usage:

```
>>> img = sample_type_13.get_latent( 'PIL' )
>>> img# doctest: +ELLIPSIS
<PIL.Image.Image image mode=L size=800x768 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'bd5641d4006e395c4e5d7459d2485891'
```

The format of the python object returned can be specified as parameter:

```
>>> raw = sample_type_13.get_latent( 'RAW' )
>>> type( raw )
<type 'str'>
```

get_latent_annotated(*idc=-1*, ***options*)

Function to return the annotated latent.

Parameters **idc** (*int*) – IDC value.

Returns Annotated fingermark

Return type PIL.Image

Usage:

```
>>> img = sample_type_13.get_latent_annotated()
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=L size=800x768 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'bd5641d4006e395c4e5d7459d2485891'
```

get_latent_diptych(*idc=-1*, ***options*)

Function to return the diptych of the latent fingermark (latent and annotated latent)

Parameters **idc** (*int*) – IDC value.

Returns Latent fingermark diptych

Return type PIL.Image

Usage:

```
>>> img = sample_type_13.get_latent_diptych()
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=RGB size=1600x768 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'366a3937ace2204c9c99ae06609ed178'
```

get_latent_hull(*idc=-1*, *linewidth=None*, ***options*)

Annotate the convex Hull on the latent image. This convex Hull is calculated based on the minutiae stored in the NIST object.

Parameters

- **idc** (*int*) – IDC value.
- **linewidth** (*int*) – Width of the convex Hull line.

Returns Latent annotated with the convex Hull

Return type PIL.Image

Usage:

```
>>> img = sample_type_13.get_latent_hull()
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=RGB size=800x768 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'4310f5bbe5987c485b91586f96d949f4'
```

get_latent_triptych(*content=None*, *idc=-1*, ***options*)

get_minutia_by_id(*id*, *format=None*, *idc=-1*)

Return a minutia based on the id

Parameters

- **id** (*str or int*) – Identifier of the minutia
- **format** (*str or list*) – Format of the minutiae to return.
- **idc** (*int*) – IDC value.

Returns Particular Minutia

Return type *Minutia* (page 41)

To get the minutiae '1':

```
>>> sample_type_9_10_14.get_minutia_by_id( "1" )
Minutia( i='1', x='21.95', y='20.3', t='101', q='00', d='D' )
```

```
>>> sample_type_9_10_14.get_minutia_by_id( 1 )
Minutia( i='1', x='21.95', y='20.3', t='101', q='00', d='D' )
```

The format can also be specified as follow:

```
>>> sample_type_9_10_14.get_minutia_by_id( "1", "xy" )
Minutia( x='21.95', y='20.3' )
```

If the IDC value is specified instead of the 'format' parameter, the format is set to the default value:

```
>>> sample_type_9_10_14.get_minutia_by_id( "1", 1 )
Minutia( i='1', x='21.95', y='20.3', t='101', q='00', d='D' )
```

If the id is not found in the NIST object, the value 'None' is returned:

```
>>> sample_type_9_10_14.get_minutia_by_id( "1337" ) == None
True
```

get_minutiae (*format=None*, *idc=-1*, ***options*)

Get the minutiae information from the field 9.012 for the IDC passed in argument.

Parameters

- **format** (*str or list*) – Format of the minutiae to return.
- **idc** (*int*) – IDC value.

Returns List of minutiae

Return type *AnnotationList* (page 34)

The parameter 'format' allow to select the data to extract:

- i: Index number
- x: X coordinate
- y: Y coordinate
- t: Angle theta
- d: Type designation
- q: Quality

The 'format' parameter is optional. The IDC value can be passed in parameter even without format. The default format ('ixytdq') will be used.

To get all information, dont speficy any format:

```
>>> sample_type_9_10_14.get_minutiae() # doctest: +NORMALIZE_WHITESPACE
[
    Minutia( i='1', x='21.95', y='20.3', t='101', q='00', d='D' ),
    Minutia( i='2', x='21.49', y='20.3', t='96', q='00', d='D' ),
    Minutia( i='3', x='18.59', y='24.0', t='96', q='00', d='D' ),
    Minutia( i='4', x='20.83', y='23.55', t='98', q='00', d='D' ),
    Minutia( i='5', x='21.44', y='23.19', t='104', q='00', d='D' ),
    Minutia( i='6', x='21.29', y='25.32', t='112', q='00', d='D' ),
    Minutia( i='7', x='19.56', y='25.53', t='112', q='00', d='D' ),
    Minutia( i='8', x='20.57', y='25.58', t='111', q='00', d='D' ),
    Minutia( i='9', x='23.32', y='23.45', t='107', q='00', d='D' ),
    Minutia( i='10', x='21.23', y='15.06', t='193', q='00', d='D' ),
    Minutia( i='11', x='17.98', y='12.22', t='201', q='00', d='D' ),
```

(continues on next page)

(continued from previous page)

```
    Minutia( i='12', x='16.76', y='12.52', t='203', q='00', d='D' ),
    Minutia( i='13', x='14.78', y='13.34', t='203', q='00', d='D' ),
    Minutia( i='14', x='11.48', y='15.01', t='210', q='00', d='D' ),
    Minutia( i='15', x='19.61', y='14.0', t='210', q='00', d='D' ),
    Minutia( i='16', x='17.73', y='12.93', t='211', q='00', d='D' ),
    Minutia( i='17', x='13.61', y='13.49', t='211', q='00', d='D' ),
    Minutia( i='18', x='14.88', y='24.21', t='218', q='00', d='D' ),
    Minutia( i='19', x='20.22', y='14.4', t='215', q='00', d='D' ),
    Minutia( i='20', x='13.61', y='14.61', t='224', q='00', d='D' ),
    Minutia( i='21', x='15.65', y='23.85', t='220', q='00', d='D' ),
    Minutia( i='22', x='18.59', y='14.81', t='224', q='00', d='D' ),
    Minutia( i='23', x='20.32', y='16.03', t='235', q='00', d='D' ),
    Minutia( i='24', x='15.09', y='17.4', t='239', q='00', d='D' ),
    Minutia( i='25', x='14.88', y='22.63', t='244', q='00', d='D' ),
    Minutia( i='26', x='15.29', y='21.82', t='243', q='00', d='D' ),
    Minutia( i='27', x='16.46', y='20.55', t='249', q='00', d='D' ),
    Minutia( i='28', x='18.29', y='20.45', t='250', q='00', d='D' ),
    Minutia( i='29', x='19.2', y='19.23', t='256', q='00', d='D' ),
    Minutia( i='30', x='18.14', y='21.36', t='262', q='00', d='D' ),
    Minutia( i='31', x='21.34', y='24.05', t='290', q='00', d='D' ),
    Minutia( i='32', x='17.53', y='22.48', t='290', q='00', d='D' ),
    Minutia( i='33', x='23.32', y='15.62', t='329', q='00', d='D' ),
    Minutia( i='34', x='23.27', y='15.16', t='350', q='00', d='D' ),
    Minutia( i='35', x='21.79', y='14.61', t='16', q='00', d='D' ),
    Minutia( i='36', x='19.2', y='13.39', t='30', q='00', d='D' ),
    Minutia( i='37', x='15.44', y='23.75', t='45', q='00', d='D' ),
    Minutia( i='38', x='16.66', y='22.68', t='56', q='00', d='D' ),
    Minutia( i='39', x='14.63', y='22.43', t='52', q='00', d='D' ),
    Minutia( i='40', x='16.1', y='19.79', t='58', q='00', d='D' ),
    Minutia( i='41', x='16.36', y='22.23', t='59', q='00', d='D' ),
    Minutia( i='42', x='15.95', y='17.96', t='57', q='00', d='D' ),
    Minutia( i='43', x='16.97', y='21.87', t='72', q='00', d='D' ),
    Minutia( i='44', x='18.9', y='20.55', t='71', q='00', d='D' ),
    Minutia( i='45', x='21.44', y='17.81', t='77', q='00', d='D' ),
    Minutia( i='46', x='20.22', y='21.41', t='85', q='00', d='D' ),
    Minutia( i='47', x='21.13', y='21.06', t='89', q='00', d='D' ),
    Minutia( i='48', x='19.71', y='22.23', t='85', q='00', d='D' )
]
```

```
>>> [ m.as_list() for m in sample_type_9_10_14.get_minutiae() ]
[['1', 21.95, 20.3, 101, '00', 'D'], ['2', 21.49, 20.3, 96, '00', 'D'], ['3
↪', 18.59, 24.0, 96, '00', 'D'], ['4', 20.83, 23.55, 98, '00', 'D'], ['5',
↪ 21.44, 23.19, 104, '00', 'D'], ['6', 21.29, 25.32, 112, '00', 'D'], ['7
↪', 19.56, 25.53, 112, '00', 'D'], ['8', 20.57, 25.58, 111, '00', 'D'], [
↪'9', 23.32, 23.45, 107, '00', 'D'], ['10', 21.23, 15.06, 193, '00', 'D'],
↪ ['11', 17.98, 12.22, 201, '00', 'D'], ['12', 16.76, 12.52, 203, '00', 'D
↪'], ['13', 14.78, 13.34, 203, '00', 'D'], ['14', 11.48, 15.01, 210, '00',
↪ 'D'], ['15', 19.61, 14.0, 210, '00', 'D'], ['16', 17.73, 12.93, 211, '00
↪', 'D'], ['17', 13.61, 13.49, 211, '00', 'D'], ['18', 14.88, 24.21, 218,
↪'00', 'D'], ['19', 20.22, 14.4, 215, '00', 'D'], ['20', 13.61, 14.61,␣
↪224, '00', 'D'], ['21', 15.65, 23.85, 220, '00', 'D'], ['22', 18.59, 14.
↪81, 224, '00', 'D'], ['23', 20.32, 16.03, 235, '00', 'D'], ['24', 15.09,␣
↪17.4, 239, '00', 'D'], ['25', 14.88, 22.63, 244, '00', 'D'], ['26', 15.
↪29, 21.82, 243, '00', 'D'], ['27', 16.46, 20.55, 249, '00', 'D'], ['28',␣
↪18.29, 20.45, 250, '00', 'D'], ['29', 19.2, 19.23, 256, '00', 'D'], ['30
↪', 18.14, 21.36, 262, '00', 'D'], ['31', 21.34, 24.05, 290, '00', 'D'], [
↪'32', 17.53, 22.48, 290, '00', 'D'], ['33', 23.32, 15.62, 329, '00', 'D
↪'], ['34', 23.27, 15.16, 350, '00', 'D'], ['35', 21.79, 14.61, 16, '00',
↪'D'], ['36', 19.2, 13.39, 30, '00', 'D'], ['37', 15.44, 23.75, 45, '00',
↪'D'], ['38', 16.66, 22.68, 56, '00', 'D'], ['39', 14.63, 22.43, 52, '00',
↪ 'D'], ['40', 16.1, 19.79, 58, '00', 'D'], ['41', 16.36, 22.23, 59, '00',
↪ 'D'], ['42', 15.95, 17.96, 57, '00', 'D'], ['43', 16.97, 21.87, 72, '00
↪', 'D'], ['44', 18.9, 20.55, 71, '00', 'D'], ['45', 21.44, 17.81, 77, '00
↪', 'D'], ['46', 20.22, 21.41, 85, '00', 'D'], ['47', 21.13, 21.06, 89,
↪'00', 'D'], ['48', 19.71, 22.23, 85, '00', 'D']]
```

(continues on next page)

The format parameter is used by the `AnnotationsList()` object to sort the fields returned.

get_minutiaeCount (*idc=-1*)

Return the number of minutiae stored in the current NIST object. This function does not cross-check the count with the content of the minutiae field; a difference can occur if manually modified.

Parameters **idc** (*int*) – IDC value.

Returns Number of minutiae

Return type int

Usage:

```
>>> sample_type_9_10_14.get_minutiaeCount()
48
>>> sample_type_9_10_14.get_minutiaeCount( idc = 1 )
48
>>> sample_type_17_iris.get_minutiaeCount()
Traceback (most recent call last):
    ...
ntypeNotFound
```

get_minutiae_all (*format=None*)

Return the minutiae for all 10 fingers. If the idc is not present in the NIST object, i.e. the finger is missing, an empty list of minutiae is returned, to complete the tenprint card.

Parameters **format** (*str or tuple*) – Format of the Minutiae to return.

Returns List of AnnotationList

Return type list

To get all minutiae for all finger stored in a NIST object:

```
>>> sample_type_9_10_14.get_minutiae_all() # doctest: +NORMALIZE_
↪WHITESPACE +ELLIPSIS
[[
    Minutia( i='1', x='21.95', y='20.3', t='101', q='00', d='D' ),
    Minutia( i='2', x='21.49', y='20.3', t='96', q='00', d='D' ),
    Minutia( i='3', x='18.59', y='24.0', t='96', q='00', d='D' ),
    Minutia( i='4', x='20.83', y='23.55', t='98', q='00', d='D' ),
    ...
    Minutia( i='45', x='21.44', y='17.81', t='77', q='00', d='D' ),
    Minutia( i='46', x='20.22', y='21.41', t='85', q='00', d='D' ),
    Minutia( i='47', x='21.13', y='21.06', t='89', q='00', d='D' ),
    Minutia( i='48', x='19.71', y='22.23', t='85', q='00', d='D' )
], [], [], [], [], [], [], [], [], []]
```

If the NIST object does not contain a Type04, Type14 or Type13 record, the function will rise an notImplemented exception:

```
>>> sample_type_17_iris.get_minutiae_all()
Traceback (most recent call last):
...
notImplemented
```

get_minutiae_by_type (*designation*, *format=None*, *idc=-1*)

Filter the minutiae list by type

Parameters

- **designation** (*str or list*) – Type designation to keep

- **format** (*str or list*) – Format of the minutiae to return.
- **idc** (*int*) – IDC value.

Returns List of minutiae

Return type *AnnotationList* (page 34)

To get only the minutiae with the type designation set as 'D' (unknown):

```
>>> sample_type_9_10_14.get_minutiae_by_type( "D" ) # doctest: +NORMALIZE_
↪WHITESPACE +ELLIPSIS
[
    Minutia( i='1', x='21.95', y='20.3', t='101', q='00', d='D' ),
    Minutia( i='2', x='21.49', y='20.3', t='96', q='00', d='D' ),
    Minutia( i='3', x='18.59', y='24.0', t='96', q='00', d='D' ),
    Minutia( i='4', x='20.83', y='23.55', t='98', q='00', d='D' ),
    ...
    Minutia( i='45', x='21.44', y='17.81', t='77', q='00', d='D' ),
    Minutia( i='46', x='20.22', y='21.41', t='85', q='00', d='D' ),
    Minutia( i='47', x='21.13', y='21.06', t='89', q='00', d='D' ),
    Minutia( i='48', x='19.71', y='22.23', t='85', q='00', d='D' )
]
```

get_palmar (*format='PIL', idc=-1, fpc=None*)

Return the palmar image, WSQ or PIL format.

Parameters

- **format** (*str*) – Format of the returned image.
- **idc** (*int*) – IDC value.

Returns Palmar image

Return type PIL.Image or str

Raises **notImplemented** – if the NIST object does not contain Type15

```
>>> img = sample_type_15_palms.get_palmar( idc = 2 )
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'f2fa02016878ff1e3e23781855f9be6c'
```

get_print (*format='PIL', idc=-1, fpc=None*)

Return the print image, WSQ or PIL format.

Parameters

- **format** (*str*) – Format of the returned image.
- **idc** (*int*) – IDC value.

Returns Print image

Return type PIL.Image or str

Raises

- **notImplemented** – if the NIST object does not contain Type04 or Type14
- **notImplemented** – if the image format is not supported

Usage:

```
>>> img = sample_type_4_tpcard.get_print( "PIL", 1 )
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=L size=804x752 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'e9ed876ab9de5ccc5b647eafd37b9162'
```

get_print_annotated(*idc=-1*)

Function to return the annotated print.

Parameters **idc** (*int*) – IDC value.

Returns Annotated fingerprint image

Return type PIL.Image

Usage:

```
>>> img = sample_type_4_tpcard.get_print_annotated( 1 )
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=RGB size=804x752 at ...>
```

get_print_diptych(*idc=-1*)

Function to return the diptych of the latent fingermark (latent and annotated latent)

Parameters **idc** (*int*) – IDC value.

Returns Fingerprint diptych (fingerprint and fingerprint annotated)

Return type PIL.Image

Usage:

```
>>> img = sample_type_9_10_14.get_print_diptych( 1 )
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=RGB size=1600x768 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'1c7c69848bf554759733c8f71eeb58d9'
```

get_resolution(*idc=-1*)

Return the (horizontal) resolution of image in DPI.

Parameters **idc** (*int*) – IDC value.

Returns Resolution in DPI

Return type int

Usage:

```
>>> sample_type_9_10_14.get_resolution()
500
```

get_size(*idc=-1*)

Get a python-tuple representing the size of the image.

Parameters **idc** (*int*) – IDC value.

Returns Horizontal and vertical size in px.

Return type tuple of int

Usage:

```
>>> sample_type_9_10_14.get_size( 1 )
(800, 768)
```

get_tenprint(*annotated=False*)

Function to return the tenprint image of the current NIST fingerprint object (only the rolled finger 1 to 10, two rows of 5 images).

Returns Tenprint image.

Return type PIL.Image

Usage:

```
>>> img = sample_type_4_tpcard.get_tenprint()
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=L size=4020x1504 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'8da4bdb447bfd07380e382e14d90453c'
```

get_tenprint_annotated()

Return the tenprint image annoated.

..see: *get_tenprint()* (page 20) function

get_tenprintcard_back(*outres=1000*)

Return the tenprint card for the palmar print. This function returns an ISO-A4 Swiss tenprint card.

Parameters **outres** (*int*) – Output resolution of the tenprint card, in DPI.

Returns Tenprint card.

Return type PIL.Image

get_tenprintcard_front(*outres=1000*)

Return the tenprint card for the rolled fingers 1 to 10. This function returns an ISO-A4 Swiss tenprint card.

Parameters **outres** (*int*) – Output resolution of the tenprint card, in DPI.

Returns Tenprint card.

Return type PIL.Image

Usage:

```
>>> img = sample_type_4_tpcard.get_tenprintcard_front()
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=L size=8268x11692 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'0eed7645e6ca37b3c8ee4106dee3f225'
```

get_width(*idc=-1*)

Return the width of the image.

Parameters **idc** (*int*) – IDC value.

Returns Width

Return type int

Usage:

```
>>> sample_type_9_10_14.get_width()
800
```

init_latent(**args*, ***kwargs*)

Initialize an latent fingermark NIST object. If the correct data is passed as keyword arguments (ie with the same names as in the *add_Type09()* (page 6), *add_Type13()* (page 6) and *set_latent()* (page 27) functions), the corresponding fields will be populated.

Returns Latent NIST object

Return type *NISTf* (page 6)

Usage:

```
>>> from NIST import NISTf
>>> from NIST.fingerprint.functions import AnnotationList
```

```
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> params = {
...     'minutiae': minutiae,
...     'cores': [ [ 12.5, 18.7 ] ]
... }
>>> mark_nist = NISTf().init_latent( **params )
```

See also:

`add_Type01()` `add_Type02()` *add_Type09()* (page 6) *add_Type13()* (page 6) *set_latent()* (page 27)

init_new(**args*, ***kwargs*)

Initialize a new latent fingermark or fingerprint NIST object. See the functions *init_latent()* (page 21) and *init_print()* (page 24) for more details.

Returns Latent or print fingerprint NIST object.

Return type *NISTf* (page 6)

Raises **notImplemented** – if the NIST object does not contain Type04, Type13 or Type14 data.

New latent fingermark:

```
>>> from NIST import NISTf
>>> from NIST.fingerprint.functions import AnnotationList
```

```
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> params = {
...     'type': 'latent',
...     'minutiae': minutiae,
...     'cores': [ [ 12.5, 18.7 ] ]
... }
>>> mark_nist = NISTf().init_latent( **params )
```

(continues on next page)

(continued from previous page)

```
>>> print( mark_nist ) # doctest: +NORMALIZE_WHITESPACE, +ELLIPSIS
Informations about the NIST object:
    Obj ID:  ...
    Records: Type-01, Type-02, Type-09, Type-13
    Class:   NISTf
<BLANKLINE>
NIST Type-01
    01.001 LEN: 00000145
    01.002 VER: 0300
    01.003 CNT: 1<US>3<RS>2<US>0<RS>9<US>0<RS>13<US>0
    01.004 TOT: USA
    01.005 DAT: ...
    01.006 PRY: 1
    01.007 DAI: FILE
    01.008 ORI: UNIL
    01.009 TCN: ...
    01.011 NSR: 00.00
    01.012 NTR: 00.00
NIST Type-02 (IDC 0)
    02.001 LEN: 00000038
    02.002 IDC: 0
    02.004    : ...
NIST Type-09 (IDC 0)
    09.001 LEN: 00000266
    09.002 IDC: 0
    09.003 IMP: 4
    09.004 FMT: S
    09.007    : U
    09.008    : 12501870
    09.010    : 10
    09.011    : 0
    09.012    : 1<US>07850705290<US>0<US>A<RS>2<US>13801530155<US>0<US>A
 ↪<RS>3<US>11462232224<US>0<US>B<RS>4<US>22612517194<US>0<US>A<RS>5<US>
 ↪06970848153<US>0<US>B<RS>6<US>12581988346<US>0<US>A<RS>7<US>19691980111
 ↪<US>0<US>C<RS>8<US>12310387147<US>0<US>A<RS>9<US>13881429330<US>0<US>D
 ↪<RS>10<US>15472249271<US>0<US>D
NIST Type-13 (IDC 0)
    13.001 LEN: 00250150
    13.002 IDC: 0
    13.003 IMP: 4
    13.004 SRC: UNIL
    13.005 LCD: ...
    13.006 HLL: 500
    13.007 VLL: 500
    13.008 SLC: 1
    13.009 THPS: 500
    13.010 TVPS: 500
    13.011 CGA: 0
    13.012 BPX: 8
    13.013 FGP: 0
    13.999 DATA: FFFFFFFF ... FFFFFFFF (250000 bytes)
```

New fingerprint:

```
>>> from NIST import NISTf
>>> from NIST.fingerprint.functions import AnnotationList
```

```
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
 ↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
 ↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
 ↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 14[...]
 ↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
 ↪format = "ixytqd", type = 'Minutia' )
```

(continues on next page)

(continued from previous page)

```
>>> params = {
...     'type': 'print',
...     'minutiae': minutiae,
...     'cores': [ [ 12.5, 18.7 ] ]
... }
>>> print_nist = NISTf().init_print( **params )
>>> print( print_nist ) # doctest: +NORMALIZE_WHITESPACE, +ELLIPSIS
Informations about the NIST object:
    Obj ID:  ...
    Records: Type-01, Type-02, Type-04, Type-09
    Class:   NISTf
<BLANKLINE>
NIST Type-01
    01.001 LEN: 00000144
    01.002 VER: 0300
    01.003 CNT: 1<US>3<RS>2<US>0<RS>4<US>1<RS>9<US>0
    01.004 TOT: USA
    01.005 DAT: ...
    01.006 PRY: 1
    01.007 DAI: FILE
    01.008 ORI: UNIL
    01.009 TCN: ...
    01.011 NSR: 19.69
    01.012 NTR: 19.69
NIST Type-02 (IDC 0)
    02.001 LEN: 00000038
    02.002 IDC: 0
    02.004    : ...
NIST Type-04 (IDC 1)
    04.001 LEN: 250018
    04.002 IDC: 1
    04.003 IMP: 3
    04.004 FGP: 0
    04.005 ISR: 0
    04.006 HLL: 500
    04.007 VLL: 500
    04.008 CGA: 0
    04.999    : FFFFFFFF ... FFFFFFFF (250000 bytes)
NIST Type-09 (IDC 0)
    09.001 LEN: 00000266
    09.002 IDC: 0
    09.003 IMP: 4
    09.004 FMT: S
    09.007    : U
    09.008    : 12501870
    09.010    : 10
    09.011    : 0
    09.012    : 1<US>07850705290<US>0<US>A<RS>2<US>13801530155<US>0<US>A
 ↪<RS>3<US>11462232224<US>0<US>B<RS>4<US>22612517194<US>0<US>A<RS>5<US>
 ↪06970848153<US>0<US>B<RS>6<US>12581988346<US>0<US>A<RS>7<US>19691980111
 ↪<US>0<US>C<RS>8<US>12310387147<US>0<US>A<RS>9<US>13881429330<US>0<US>D
 ↪<RS>10<US>15472249271<US>0<US>D
```

init_print (**args*, ***kwargs*)

Initialize an fingerprint NIST object. If the correct data is passed as keyword arguments (ie with the same names as in the *add_Type04()* (page 6), *add_Type09()* (page 6) and *set_print()* (page 28) functions), the corresponding fields will be populated.

Returns Print NIST object

Return type *NISTf* (page 6)

Usage:

```
>>> from NIST import NISTf
>>> from NIST.fingerprint.functions import AnnotationList
```

```
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> params = {
...     'minutiae': minutiae,
...     'cores': [ [ 12.5, 18.7 ] ]
... }
>>> print_nist = NISTf().init_print( **params )
```

See also:

`add_Type01()` `add_Type02()` *add_Type04()* (page 6) *set_print()* (page 28) *add_Type09()* (page 6)

migrate_Type04_to_Type14()

Migration of the Type04 fingerprint record to Type14 record. This function make a copy of the fingerprint images informations. The minutiae are not modified. The Type04 records are deleted after conversion.

The Type14 fingerprint are stored in RAW format. The WSQ compression is not supported for the moment.

Usage:

```
>>> sample_type_4_tpcard.migrate_Type04_to_Type14()
```

mm2px(*data*, *idc=-1*)

Transformation the coordinates from pixel to millimeters

Parameters

- **data** (*tuple*) – Coordinate value.
- **idc** (*int*) – IDC value.

Usage:

```
>>> sample_type_13.mm2px( ( 12.7, 12.7 ) )
[250.0, 250.0]
```

patch_to_standard()

Check some requirements for the NIST file. Fields checked:

- 1.011 and 1.012
- 4.005
- 9.004

This function call the `NIST.traditional.NIST.patch_to_standard()` function afterward.

process_minutiae_field(*minutiae*, *field*, *idc=-1*)

Internal function to convert the stored minutiae to a usable format. This function should not be used

directly (but can be if you really want). The format of the minutiae is automatically detected and process accordingly.

px2mm(*data*, *idc=-1*)

Transformation the coordinates from pixels to millimeters

Parameters

- **data** (*tuple*) – Coordinate value.
- **idc** (*int*) – IDC value.

Usage:

```
>>> sample_type_13.px2mm( ( 250.0, 250.0 ) )
[12.7, 12.7]
```

set_cores(*data*, *idc=-1*)

Set the core position in field 9.008. The data passed in parameter can be a single core position, or a list of cores (the cores will be stored in the field 9.008, separated by a RS separator).

Parameters

- **data** (*list*) – List of cores coordinates
- **idc** (*int*) – IDC value.

Returns If the value have been set (without double check).

Return type boolean

The cores can be set with a simple list (or tuple):

```
>>> sample_type_9_10_14.set_cores( [ 10.0, 12.7 ] )
True
>>> sample_type_9_10_14.get_cores() # doctest: +NORMALIZE_WHITESPACE
[
    Core( x='10.0', y='12.7' )
]
```

A list of lists (for multiples cores):

```
>>> sample_type_9_10_14.set_cores( [ [ 12.5, 18.7 ], [ 10.0, 12.7 ] ], 1 )
True
>>> sample_type_9_10_14.get_cores( 1 ) # doctest: +NORMALIZE_WHITESPACE
[
    Core( x='12.5', y='18.7' ),
    Core( x='10.0', y='12.7' )
]
```

With an AnnotationList object:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> cores = AnnotationList()
>>> cores.from_list( [ [ 12.5, 18.7 ], [ 10.0, 12.7 ] ], format = "xy",␣
↪type = 'Core' )
>>> cores # doctest: +NORMALIZE_WHITESPACE
[
    Core( x='12.5', y='18.7' ),
    Core( x='10.0', y='12.7' )
]
>>> sample_type_9_10_14.set_cores( cores, 1 )
True
>>> sample_type_9_10_14.get_cores( 1 ) # doctest: +NORMALIZE_WHITESPACE
[
    Core( x='12.5', y='18.7' ),
```

(continues on next page)

(continued from previous page)

```
    Core( x='10.0', y='12.7' )
]
```

If no data is passed to the function, 'False' is returned:

```
>>> sample_type_9_10_14.set_cores( None, 1 )
False
>>> sample_type_9_10_14.set_cores( [], 1 )
False
```

If the format is not supported, the functions raises an formatNotSupported Exception:

```
>>> sample_type_9_10_14.set_cores( "sample/cores.txt", 1 )
Traceback (most recent call last):
...
formatNotSupported
```

set_height (*ntype*, *value*, *idc=-1*)

Set the image height for the ntype specified in parameter.

Parameters

- **ntype** (*int*) – ntype to set the image size.
- **value** (*int or str*) – height of the image.
- **idc** (*int*) – IDC value.

Raises notImplemented – if the NIST object does not contain Type04, Type13 or Type14 data.

Usage:

```
>>> sample_type_13.set_height( 13, 500 )
>>> sample_type_13.get_field( "13.007" )
'500'
```

set_latent (*image=None*, *res=500*, *idc=-1*, ***options*)

Detect the type of image passed in parameter and store it in the 13.999 field. If no image is passed in argument, an empty image is set in the NIST object.

Parameters

- **image** (*PIL.Image or str*) – Input image to store in the NIST object.
- **res** (*int*) – Image resolution in DPI.
- **idc** (*int*) – IDC value.

Set an PIL.Image image:

```
>>> from PIL import Image
>>> image = Image.new( "L", ( 800, 768 ), 255 )
>>> sample_type_13.set_latent( image )
```

```
>>> img = sample_type_13.get_latent()
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=L size=800x768 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'23f9d5b5ad8e962ed093fe287e8e5276'
```

Set an string image (RAW format):

```
>>> w, h = sample_type_13.get_size()
>>> sample_type_13.set_latent( chr( 255 ) * w * h )
```

```
>>> img = sample_type_13.get_latent()
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=L size=800x768 at ...>
```

set_latent_size (*value*, *idc=-1*)
Set the size of the latent image.

Parameters

- **value** (*tuple*) – Size of the image (width, height)
- **idc** (*int*) – IDC value.

Usage:

```
>>> sample_type_13.set_latent_size( ( 500, 500 ) )
```

set_minutiae (*data*, *idc=-1*)
Set the minutiae in the field 9.012.

Parameters

- **data** (AnnotationList (page 34) *or str*) – List of minutiae coordinates
- **idc** (*int*) – IDC value.

Returns Number of minutiae added to the NIST object

Return type int

Raises **minutiaeFormatNotSupported** – if the format is not supported

Usage:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
>>> sample_type_9_10_14.set_minutiae( minutiae, 1 )
10
```

The parameter 'data' can be a list or an AnnotationList. Otherwise, the function will raise a minutiaeFormatNotSupported Exception.

```
>>> sample_type_9_10_14.set_minutiae( [ 12, 13, 14 ], 1 )
Traceback (most recent call last):
...
minutiaeFormatNotSupported
```

set_print (*image=None*, *res=None*, *size=(512, 512)*, *format='RAW'*, *idc=-1*, ***options*)
Function to set an print image to the 4.999 field, and set the size.

Parameters

- **image** (*PIL.Image or str*) – Image to set in the NIST object.
- **res** (*int*) – Resolution of the image in DPI.
- **size** (*tuple*) – Size of the image.
- **format** (*str*) – Format of the image (WSQ or RAW).

- **idc** (*int*) – IDC value

Usage:

```
>>> from PIL import Image
>>> image = Image.new( "L", ( 500, 500 ), 255 )
>>> sample_type_4_tpcard.set_print( image, format = "RAW", idc = 1 )
>>> img = sample_type_4_tpcard.get_print( idc = 1 )
>>> img # doctest: +ELLIPSIS
<PIL.Image.Image image mode=L size=500x500 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'7157c3d901362236afbdd84de3f61007'
```

set_print_size (*value*, *idc=-1*)

Set the size of the fingerprint image.

Parameters

- **value** (*tuple*) – Size of the image (width, height).
- **idc** (*int*) – IDC value.

Raises **notImplemented** – if the NIST object does not contain Type04 or Type14 data

Usage:

```
>>> sample_type_4_tpcard.set_print_size( ( 500, 500 ), 1 )
```

set_resolution (*res*, *idc=-1*)

Set the resolution in DPI.

Parameters **idc** (*int*) – IDC value.

Usage:

```
>>> sample_type_9_10_14.set_resolution( 500 )
>>> sample_type_9_10_14.get_resolution()
500
```

set_size (*value*, *idc=-1*)

Set the image size (width, height) for the ntype specified in parameter.

Parameters

- **ntype** (*int*) – ntype to set the image size.
- **value** (*int or str*) – Siez (width, height) of the image.
- **idc** (*int*) – IDC value.

Raises **notImplemented** – if the NIST object does not contain Type04, Type13 or Type14 data.

Usage:

```
>>> sample_type_13.set_size( ( 500, 500 ) )
>>> sample_type_13.get_height()
500
>>> sample_type_13.get_width()
500
```

set_width (*ntype*, *value*, *idc=-1*)

Set the image width for the ntype specified in parameter.

Parameters

- **ntype** (*int*) – ntype to set the image size.
- **value** (*int or str*) – width of the image.
- **idc** (*int*) – IDC value.

Raises **notImplemented** – if the NIST object does not contain Type04, Type13 or Type14 data.

Usage:

```
>>> sample_type_13.set_width( 13, 500 )
>>> sample_type_13.get_field( "13.006" )
'500'
```

Misc functions

class NIST.fingerprint.functions.**Annotation**(**args*, ***kwargs*)

Bases: object

Annotation Class; generic class for Minutia and Core information. This class is not designed to be directly used, but should be overloaded with some custom class (see *Minutia()* (page 41) and *Core()* (page 40) for more details).

Variables ***defaultformat*** (page 41) – Default format to store the Annotation data.

Import:

```
>>> from NIST.fingerprint.functions import Annotation
```

Usage:

```
>>> Annotation( [ 1.0, 2.1, 3.18 ], format = "abc" )
Annotation( a='1.0', b='2.1', c='3.18' )
```

By default, if the data is not provided, None is returned:

```
>>> Annotation( [ 1.0, 2.1, 3.18 ], format = "abcd" )
Annotation( a='1.0', b='2.1', c='3.18', d='None' )
```

The Annotation can also be initialized with keyword arguments:

```
>>> Annotation( a = 1.0, b = 2.1, c = 3.18 )
Annotation( a='1.0', c='3.18', b='2.1' )
```

Note: Since the keyword arguments are stored in a dictionary, the order of the keys are not ensured. This will be corrected in Python3.6.

```
>>> Annotation( c = 3.18, b = 2.1, a = 1.0 )
Annotation( a='1.0', c='3.18', b='2.1' )
```

__eq__(*other*)

Compare the Annotation object with all variables used in the _format string. The hidden data is not used while comparing the objects. The order of the variables is not important. The comparison is made regardess of the type of the data stored.

Usage :

```
>>> from NIST.fingerprint.functions import Annotation
>>> a = Annotation( [ 1, 2, 3 ], format = "xyt" )
>>> b = Annotation( [ 1, 3, 2 ], format = "xty" )
```

(continues on next page)

(continued from previous page)

```
>>> c = Annotation( [ 1, 2, 3.0 ], format = "xyt" )
>>> d = Annotation( [ 1, 2, 3.0 ], format = "xyt" )
>>> d.id = 18
```

```
>>> e = Annotation( [ 1, 2, 4 ], format = "xyt" )
>>> f = Annotation( [ 1, 2 ], format = "xy" )
```

```
>>> a == b
True
>>> a == c
True
>>> a == d
True
```

```
>>> a == e
False
>>> a == f
False
```

__getattr__(*name*)

Get a value stored in the Annotation object. If the value is not stored in the Annotation object, the 'None' value is returned.

Let 'a' be defined as follow:

```
>>> from NIST.fingerprint.functions import Annotation
>>> a = Annotation( [ 1, 2, 3 ], format = "xyt" )
```

To retrieve the variable 'x', use the following command:

```
>>> a.x
1
```

Note: The following instructions are equivalent:

- a.var <=> a.__getattr__(var)
- a.var <=> Annotation.__getattr__(a, var)

If the variable name is stored in a python variable (dynamic access), the value can be retrived as follow:

```
>>> var = 'x'
>>> a.__getattr__( var )
1
```

__getitem__(*index*)

Function to get an item stored in the Annotation object like in a *list* object. If the variable is not present in the _data object, 'None' is returned.

Parameters **index** (*str or int*) – Index to retrieve

Let 'a' be defined as follow:

```
>>> from NIST.fingerprint.functions import Annotation
>>> a = Annotation( [ 1.0, 2.1, 3.18 ], format = "abc" )
```

To retrieve the variable 'a', you can either call it with the variable name:

```
>>> a[ 'a' ]
1.0
```

or use the index in the Annotation object (here, the fist element, ie the index 0):

```
>>> a[ 0 ]
1.0
```

__iadd__(*delta*)

Overload of the '+=' operator, allowing to offset an Annotation with a tuple (dx, dy).

Parameters **delta** (*tuple*) – Offset to apply

Let 'a' be defined as follow:

```
>>> from NIST.fingerprint.functions import Annotation
>>> a = Annotation( [ 1, 2, 3 ], format = "xyt" )
>>> a
Annotation( x='1', y='2', t='3' )
```

To shift the Annotation by a distance *offset*, use the following commands.

```
>>> offset = ( 10, 12 )
>>> a += offset
>>> a
Annotation( x='11', y='14', t='3' )
```

Note: This function works only for (x, y) coordinates. The other variables stores in the *Annotation* object are not changed.

Note: The following instructions are equivalent:

- a += delta <=> a = a.__iadd__(delta)
- a += delta <=> a = Annotation.__iadd__(a, delta)

__init__(**args*, ***kwargs*)

Constructor of the Annotation class. Try to feed the _data variable with the first argument passed in the __init__() function.

__iter__()

Overloading of the *__iter__()* (page 32) function, getting the information directly in the _data variable.

__len__()

Get the number of elements to except in the Annotation object (not the effective number of objects stored in this particular object).

Let 'a' be defined as follow:

```
>>> from NIST.fingerprint.functions import Annotation
>>> a = Annotation( [ 1, 2, 3 ], format = "xyt" )
```

If the format if changed as follow:

```
>>> a.set_format( format = "xy" )
```

then, the excepted length of the Annotation *a* is:

```
>>> len( a )
2
>>> a
Annotation( x='1', y='2' )
```

even if the data stored is not changed:

```
>>> a._data
OrderedDict([('x', 1), ('y', 2), ('t', 3)])
```

__module__ = 'NIST.fingerprint.functions'

__repr__(**args*, ***kwargs*)
Object representation. This function call the *__str__()* (page 33) function

__rshift__(*p*)
Overload of the '>>' operator to calculate the distance between two Annotations (euclidean distance). The couple (x, y) is used as coordinates.

Usage:

```
>>> from NIST.fingerprint.functions import Annotation
```

```
>>> a = Annotation( ( 1, 2 ), format = "xy" )
>>> b = Annotation( ( 2, 3 ), format = "xy" )
>>> a >> b
1.4142135623730951
```

__setattr__(*name*, *value*)
Function to set an attribute in the Annotation object.

Let 'a' be defined as follow:

```
>>> from NIST.fingerprint.functions import Annotation
>>> a = Annotation( [ 1, 2, 3 ], format = "xyt" )
```

To change the value of a variable, let say 'x', use the following command:

```
>>> a.x = 18
>>> a
Annotation( x='18', y='2', t='3' )
```

Note: All non-related object variables have to start with '_'.

The privates variables 'format' is defined (by the *set_format()* (page 34)) as follow:

```
>>> a._format = "xyt"
>>> a
Annotation( x='18', y='2', t='3' )
```

This variable is not related to the Annotation data (ie x, y and t), but have to be store in the Annotation object. All privates variables are not shone in the string representation.

__str__()
String representation of an Annotation object. Used by the print function.

The string representation is the following:

```
Annotation( var1='value1', var2='value2', ... )
```

as_json()
Return the current object as a json string.

Returns Json representation of the current Annotation

Return type str

Usage:

```
>>> from NIST.fingerprint.functions import Annotation
>>> a = Annotation( [ 1.0, 2.1, 3.18 ], format = "abc" )
>>> a.as_json()
'{"a": 1.0, "b": 2.1, "c": 3.18}'
```

as_list (*format=None*)
Return a list version of the Annotation object.

Parameters **format** (*str or list*) – Format to return

Returns List version of the Annotation object.

Return type list

Usage:

```
>>> from NIST.fingerprint.functions import Annotation
>>> a = Annotation( [ 1.0, 2.1, 3.18 ], format = "abc" )
>>> a.as_list()
[1.0, 2.1, 3.18]
>>> a.as_list( "ab" )
[1.0, 2.1]
```

defaultformat = 'i'

get (*name*, *default=None*)
Function to get a particular variable inside the Annotation object.

Let 'a' be defined as follow:

```
>>> from NIST.fingerprint.functions import Annotation
>>> a = Annotation( [ 1.0, 2.1, 3.18 ], format = "abc" )
```

To retrieve the variable 'a', you can either call it with the variable name:

```
>>> a.get( 'a' )
1.0
```

set_format (*format=None*, ***kwargs*)
Set the format in the _format variable.

Parameters **format** (*str or list*) – Format of the Annotation object

class NIST.fingerprint.functions.**AnnotationList** (*data=None*)
Bases: MDmisc.eobject.eobject

AnnotationList class; generic class to store a list of Annotation objects. The functions implemented in the AnnotationList class are (generally) a wrapper function applied to all Annotation objects stored in the AnnotationList object.

__getitem__ (*index*)
Function to get a specific Annotation from the AnnotationList object.

Parameters **index** (*int*) – Index of the value to retrive.

__iadd__ (*delta*)
Function to call the *__iadd__()* function of each element in the AnnotationList object, ie shifting all Annotations by a value of *delta*.

Parameters **delta** (*tuple*) – Offset to apply

See *NIST.fingerprint.functions.Annotation.__iadd__()* (page 32) for more details.

__init__ (*data=None*)
Initialization of the AnnotationList object, and set the data if provided.

Parameters **data** (*list of Annotation*) – Input data

__iter__()
Return an generator function over the data contained in the list.

Returns Data contained in the AnnotationList

Return type generator

__len__()
Get the number of Annotations in the AnnotationList.

Returns Number of Annotations in the AnnotataionList object.

Return type int

Usage:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> len( minutiae )
10
```

__module__ = 'NIST.fingerprint.functions'

__repr__(**args*, ***kwargs*)
Object representation of the AnnotationList. Call the *__str__()* (page 35) function.

__setitem__(*key*, *value*)
Set a value in the AnnotationList object.

Parameters

- **key** – Key to set.
- **value** (*Annotation* (page 30)) – Value to set.

Type int

__str__()
Function to generate a string representation of the AnnotationList object. This string is similar to the *pprint.pprint* function:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> minutiae # doctest: +NORMALIZE_WHITESPACE
[
    Minutia( i='1', x='7.85', y='7.05', t='290', q='0', d='A' ),
    Minutia( i='2', x='13.8', y='15.3', t='155', q='0', d='A' ),
    Minutia( i='3', x='11.46', y='22.32', t='224', q='0', d='B' ),
    Minutia( i='4', x='22.61', y='25.17', t='194', q='0', d='A' ),
    Minutia( i='5', x='6.97', y='8.48', t='153', q='0', d='B' ),
```

(continues on next page)

(continued from previous page)

```
    Minutia( i='6', x='12.58', y='19.88', t='346', q='0', d='A' ),
    Minutia( i='7', x='19.69', y='19.8', t='111', q='0', d='C' ),
    Minutia( i='8', x='12.31', y='3.87', t='147', q='0', d='A' ),
    Minutia( i='9', x='13.88', y='14.29', t='330', q='0', d='D' ),
    Minutia( i='10', x='15.47', y='22.49', t='271', q='0', d='D' )
]
```

append(*value*)

Function to append an element at the end of the AnnotationList.

Parameters **value** (Annotation (page 30)) – Annotation to add to the AnnotationList

Let a the objects *a* and *tmp* be defined as follow:

```
>>> from NIST.fingerprint.functions import AnnotationList
```

```
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> from NIST.fingerprint.functions import Minutia
```

```
>>> a = Minutia( [ 11, 22.67, 1.49, 325, 0, 'A' ], format = 'ixytqd' )
>>> tmp = minutiae.get_by_type( 'A' )
```

Then:

```
>>> tmp.append( a )
>>> tmp # doctest: +NORMALIZE_WHITESPACE
[
    Minutia( i='1', x='7.85', y='7.05', t='290', q='0', d='A' ),
    Minutia( i='2', x='13.8', y='15.3', t='155', q='0', d='A' ),
    Minutia( i='4', x='22.61', y='25.17', t='194', q='0', d='A' ),
    Minutia( i='6', x='12.58', y='19.88', t='346', q='0', d='A' ),
    Minutia( i='8', x='12.31', y='3.87', t='147', q='0', d='A' ),
    Minutia( i='11', x='22.67', y='1.49', t='325', q='0', d='A' )
]
```

as_json()

Return the current object as a json string.

Returns Json representation of the current AnnotationList

Return type str

Usage:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> minutiae.as_json()
'[ {"i": 1, "x": 7.85, "y": 7.05, "t": 290, "q": 0, "d": "A"}, {"i": 2, "x
↪": 13.8, "y": 15.3, "t": 155, "q": 0, "d": "A"}, {"i": 3, "x": 11.46, "y
↪": 22.32, "t": 224, "q": 0, "d": "B"}, {"i": 4, "x": 22.61, "y": 25.17,
↪"t": 194, "q": 0, "d": "A"}, {"i": 5, "x": 6.97, "y": 8.48, "t": 153, "q
↪": 0, "d": "B"}, {"i": 6, "x": 12.58, "y": 19.88, "t": 346, "q": 0, "d":
↪"A"}, {"i": 7, "x": 19.69, "y": 19.8, "t": 111, "q": 0, "d": "C"}, {"i":␣
↪8, "x": 12.31, "y": 3.87, "t": 147, "q": 0, "d": "A"}, {"i": 9, "x": 13.
↪88, "y": 14.29, "t": 330, "q": 0, "d": "D"}, {"i": 10, "x": 15.47, "y":␣
↪22.49, "t": 271, "q": 0, "d": "D"} ]'
```

as_list()

Return the current object data as list.

Returns List of Annotations

Return type list

Usage:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> minutiae.as_list()
[[1, 7.85, 7.05, 290, 0, 'A'], [2, 13.8, 15.3, 155, 0, 'A'], [3, 11.46, 22.
↪32, 224, 0, 'B'], [4, 22.61, 25.17, 194, 0, 'A'], [5, 6.97, 8.48, 153, 0,
↪ 'B'], [6, 12.58, 19.88, 346, 0, 'A'], [7, 19.69, 19.8, 111, 0, 'C'], [8,
↪ 12.31, 3.87, 147, 0, 'A'], [9, 13.88, 14.29, 330, 0, 'D'], [10, 15.47,␣
↪22.49, 271, 0, 'D']]
```

from_list(*data*, *format=None*, *type='Annotation'*)

Load the data from a list of lists.

Parameters

- **data** (*list of lists*) – Data to load in the AnnotationList object.
- **format** (*str*) – Format of the Annotations to return.
- **type** (*str*) – Type of Annotations to store in the AnnotationList (Annotation, Minutia, Core).

Usage:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> lst = [
...     [  1, 7.85, 7.05, 290, 0, 'A' ],
...     [  2, 13.80, 15.30, 155, 0, 'A' ],
...     [  3, 11.46, 22.32, 224, 0, 'B' ],
...     [  4, 22.61, 25.17, 194, 0, 'A' ],
...     [  5, 6.97, 8.48, 153, 0, 'B' ],
...     [  6, 12.58, 19.88, 346, 0, 'A' ],
...     [  7, 19.69, 19.80, 111, 0, 'C' ],
...     [  8, 12.31, 3.87, 147, 0, 'A' ],
...     [  9, 13.88, 14.29, 330, 0, 'D' ],
...     [ 10, 15.47, 22.49, 271, 0, 'D' ]
... ]
>>> minutiae = AnnotationList()
```

(continues on next page)

(continued from previous page)

```
>>> minutiae.from_list( lst, format = 'ixytqd', type = 'Minutia' )
>>> minutiae # doctest: +NORMALIZE_WHITESPACE
[
    Minutia( i='1', x='7.85', y='7.05', t='290', q='0', d='A' ),
    Minutia( i='2', x='13.8', y='15.3', t='155', q='0', d='A' ),
    Minutia( i='3', x='11.46', y='22.32', t='224', q='0', d='B' ),
    Minutia( i='4', x='22.61', y='25.17', t='194', q='0', d='A' ),
    Minutia( i='5', x='6.97', y='8.48', t='153', q='0', d='B' ),
    Minutia( i='6', x='12.58', y='19.88', t='346', q='0', d='A' ),
    Minutia( i='7', x='19.69', y='19.8', t='111', q='0', d='C' ),
    Minutia( i='8', x='12.31', y='3.87', t='147', q='0', d='A' ),
    Minutia( i='9', x='13.88', y='14.29', t='330', q='0', d='D' ),
    Minutia( i='10', x='15.47', y='22.49', t='271', q='0', d='D' )
]
```

get (*format=None*)

Get a copy of the current object with a specific format.

Parameters **format** (*str or lst*) – Format of the AnnotationList to return.

Returns A new AnnotationList.

Return type *AnnotationList* (page 34)

Usage:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> tmp = minutiae.get()
>>> tmp == minutiae
False
```

Note: Since the function return a copy of the current object, the two objects are separate in memory, allowing to modify one and not the other (reason why *minutiae2* is not equal to *minutiae*, even if the content is the same).

---

get_by_type (*designation*, *format=None*)

Filter the content of the AnnotationList by type designation.

Parameters

- **designation** (*str or list*) – Type designation to filter upon
- **format** (*str or lst*) – Format of the AnnotationList to return

Usage:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> minutiae.get_by_type( 'D' ) # doctest: +NORMALIZE_WHITESPACE
[
    Minutia( i='9', x='13.88', y='14.29', t='330', q='0', d='D' ),
    Minutia( i='10', x='15.47', y='22.49', t='271', q='0', d='D' )
]
```

get_format()
Get the format of the first Annotation in the AnnotationList

Returns List of values

Return type list

get_n_closest_from_point(*n*, *p*)
Return the n closest Annotations from a particular point p.

Parameters

- **n** (*int*) – Number of points to return
- **p** (Annotation (page 30)) – Point

get_n_furthest_from_point(*n*, *p*)
Return the n closest Annotations from a particular point p.

Parameters

- **n** (*int*) – Number of points to return
- **p** (Annotation (page 30)) – Point

remove(*value*)
Function to remove a particular Annotation (by value).

Usage:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> from NIST.fingerprint.functions import Minutia
>>> r = Minutia( [ 8, 12.31, 3.87, 147, 0, 'A' ], format = "ixytqd" )
>>> minutiae2 = minutiae.get()
>>> minutiae2.remove( r )
>>> minutiae2 # doctest: +NORMALIZE_WHITESPACE
[
    Minutia( i='1', x='7.85', y='7.05', t='290', q='0', d='A' ),
    Minutia( i='2', x='13.8', y='15.3', t='155', q='0', d='A' ),
    Minutia( i='3', x='11.46', y='22.32', t='224', q='0', d='B' ),
    Minutia( i='4', x='22.61', y='25.17', t='194', q='0', d='A' ),
    Minutia( i='5', x='6.97', y='8.48', t='153', q='0', d='B' ),
    Minutia( i='6', x='12.58', y='19.88', t='346', q='0', d='A' ),
    Minutia( i='7', x='19.69', y='19.8', t='111', q='0', d='C' ),
    Minutia( i='9', x='13.88', y='14.29', t='330', q='0', d='D' ),
    Minutia( i='10', x='15.47', y='22.49', t='271', q='0', d='D' )
]
```

set_format(*format*)
Change the format for all Annotations stored in the AnnotationList object.

Parameters **format** (*str*) – Format to change to.

Usage :

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> minutiae.set_format( 'xy' )
>>> minutiae # doctest: +NORMALIZE_WHITESPACE
[
    Minutia( x='7.85', y='7.05' ),
    Minutia( x='13.8', y='15.3' ),
    Minutia( x='11.46', y='22.32' ),
    Minutia( x='22.61', y='25.17' ),
    Minutia( x='6.97', y='8.48' ),
    Minutia( x='12.58', y='19.88' ),
    Minutia( x='19.69', y='19.8' ),
    Minutia( x='12.31', y='3.87' ),
    Minutia( x='13.88', y='14.29' ),
    Minutia( x='15.47', y='22.49' )
]
```

sort_dist_point(*p*)

Sort inplace the AnnotationList regarding the distance with a particular point p.

Usage:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> minutiae = AnnotationList()
>>> minutiae.from_list( [[ 1, 7.85, 7.05, 290, 0, 'A' ], [ 2, 13.80, 15.30,
↪ 155, 0, 'A' ], [ 3, 11.46, 22.32, 224, 0, 'B' ], [ 4, 22.61, 25.17, 194,
↪ 0, 'A' ], [ 5, 6.97, 8.48, 153, 0, 'B' ], [ 6, 12.58, 19.88, 346, 0, 'A
↪' ], [ 7, 19.69, 19.80, 111, 0, 'C' ], [ 8, 12.31, 3.87, 147, 0, 'A' ],␣
↪[ 9, 13.88, 14.29, 330, 0, 'D' ], [ 10, 15.47, 22.49, 271, 0, 'D' ]],␣
↪format = "ixytqd", type = 'Minutia' )
```

```
>>> from NIST.fingerprint.functions import Point
```

```
>>> p = Point( ( 12, 12 ) )
>>> minutiae.sort_dist_point( p )
>>> minutiae.get( "xy" ) # doctest: +NORMALIZE_WHITESPACE
[
    Minutia( x='13.88', y='14.29' ),
    Minutia( x='13.8', y='15.3' ),
    Minutia( x='6.97', y='8.48' ),
    Minutia( x='7.85', y='7.05' ),
    Minutia( x='12.58', y='19.88' ),
    Minutia( x='12.31', y='3.87' ),
    Minutia( x='11.46', y='22.32' ),
    Minutia( x='19.69', y='19.8' ),
    Minutia( x='15.47', y='22.49' ),
    Minutia( x='22.61', y='25.17' )
]
```

class NIST.fingerprint.functions.**Core**(**args*, ***kwargs*)

Bases: *NIST.fingerprint.functions.Annotation* (page 30)

Overload of the *NIST.fingerprint.functions.Annotation()* (page 30) class.

__module__ = 'NIST.fingerprint.functions'

defaultformat = 'xy'

class NIST.fingerprint.functions.**Delta**(**args*, ***kwargs*)
Bases: *NIST.fingerprint.functions.Annotation* (page 30)

Overload of the *NIST.fingerprint.functions.Annotation()* (page 30) class.

__module__ = 'NIST.fingerprint.functions'

defaultformat = 'xyabc'

class NIST.fingerprint.functions.**Minutia**(**args*, ***kwargs*)
Bases: *NIST.fingerprint.functions.Annotation* (page 30)

Overload of the *NIST.fingerprint.functions.Annotation()* (page 30) class.

__add__(*b*)

__module__ = 'NIST.fingerprint.functions'

__sub__(*b*)

default_values(*field*)

defaultformat = 'ixytqd'

class NIST.fingerprint.functions.**Point**(**args*, ***kwargs*)
Bases: *NIST.fingerprint.functions.Annotation* (page 30)

__module__ = 'NIST.fingerprint.functions'

defaultformat = 'xy'

NIST.fingerprint.functions.**changeFormatImage**(*input*, *outformat*, ***options*)
Function to change the format of the input image.

Usage:

```
>>> from NIST.fingerprint.functions import changeFormatImage
>>> from hashlib import md5
>>> from PIL import Image
>>> from MDmisc.binary import string_to_hex
```

To convert an PIL image to a RAW string, use the following commands:

```
>>> imgPIL = Image.new( "L", ( 500, 500 ), 255 )
>>> imgRAW = changeFormatImage( imgPIL, "RAW" )
```

```
>>> md5( imgPIL.tobytes() ).hexdigest()
'7157c3d901362236afbdd84de3f61007'
>>> md5( imgRAW ).hexdigest()
'7157c3d901362236afbdd84de3f61007'
```

All format supported by PIL are supported as output format:

```
>>> changeFormatImage( imgPIL, "TIFF" ) # doctest: +ELLIPSIS
<PIL.TiffImagePlugin.TiffImageFile image mode=L size=500x500 at ...>
>>> changeFormatImage( imgPIL, "PNG" ) # doctest: +ELLIPSIS
<PIL.PngImagePlugin.PngImageFile image mode=L size=500x500 at ...>
```

You can also convert a StringIO buffer:

```
>>> from cStringIO import StringIO
>>> imgBuffer = StringIO()
>>> imgPIL.save( imgBuffer, 'JPEG' )
>>> imgRAW2 = changeFormatImage( imgBuffer, "RAW" )
```

(continues on next page)

(continued from previous page)

```
>>> imgRAW == imgRAW2
True
```

A *notImplemented* is raised if the input format is not supported or if the output format is not implemented in this function or by PIL:

```
>>> changeFormatImage( None, "RAW" )
Traceback (most recent call last):
...
notImplemented: Input format not supported
```

```
>>> d = changeFormatImage( imgPIL, "WSQ" )
>>> string_to_hex( d[ 0:4 ] )
'FFA0FFA8'
```

```
>>> md5( d ).hexdigest()
'8879e56b34aa878dd31f72b5e850d808'
```

class NIST.fingerprint.functions.**dMinutia**(**args*, ***kwargs*)

Bases: *NIST.fingerprint.functions.Annotation* (page 30)

__module__ = 'NIST.fingerprint.functions'

default_values(*field*)

defaultformat = ['dx', 'dy', 'dt']

NIST.fingerprint.functions.**diptych**(*mark*, *pr*, *markidc=-1*, *pridc=-1*)

NIST.fingerprint.functions.**lstTo012**(*lst*, *format=None*)

Convert the entire minutiae-table to the 9.012 field format.

Parameters

- **lst** (*list of lists*) – List of minutiae
- **format** (*str*) – Format of the minutiae

Returns 9.012 field

Return type str

Usage:

```
>>> from NIST.fingerprint.functions import lstTo012
>>> lstTo012(
...     [[  1,  7.85,  7.05, 290, 0, 'A' ],
...      [  2, 13.80, 15.30, 155, 0, 'A' ],
...      [  3, 11.46, 22.32, 224, 0, 'A' ],
...      [  4, 22.61, 25.17, 194, 0, 'A' ],
...      [  5,  6.97,  8.48, 153, 0, 'A' ],
...      [  6, 12.58, 19.88, 346, 0, 'A' ],
...      [  7, 19.69, 19.80, 111, 0, 'A' ],
...      [  8, 12.31,  3.87, 147, 0, 'A' ],
...      [  9, 13.88, 14.29, 330, 0, 'A' ],
...      [ 10, 15.47, 22.49, 271, 0, 'A' ]]
... )

↪'1\x1f07850705290\x1f0\x1fA\x1e2\x1f13801530155\x1f0\x1fA\x1e3\x1f11462232224\x1f0\x1fA\x1e
↪'
```

The conversion can be done with a list of (x, y, theta) coordinates. The quality will be set to '00' (expert) and the type to 'A' (Ridge ending) for compatibility with most of the AFIS (the type 'D' (Type undetermined) is not always well supported).

```
>>> lstTo012(
...     [[  7.85,  7.05, 290 ],
...      [ 13.80, 15.30, 155 ],
...      [ 11.46, 22.32, 224 ],
...      [ 22.61, 25.17, 194 ],
...      [  6.97,  8.48, 153 ],
...      [ 12.58, 19.88, 346 ],
...      [ 19.69, 19.80, 111 ],
...      [ 12.31,  3.87, 147 ],
...      [ 13.88, 14.29, 330 ],
...      [ 15.47, 22.49, 271 ]],
...     format = "xyt"
... )

↪'1\x1f07850705290\x1f00\x1fA\x1e2\x1f13801530155\x1f00\x1fA\x1e3\x1f11462232224\x1f00\x1fA\
↪'
```

```
>>> lstTo012( [] )
''
```

This function can not be called with no parameter:

```
>>> lstTo012( None )
Traceback (most recent call last):
...
notImplemented
```

NIST.fingerprint.functions.**lstTo137**(*lst*, *res=None*)

Convert the entire minutiae-table to the 9.137 field format.

Parameters

- **lst** (*list of lists*) – List of minutiae
- **format** (*str*) – Format of the minutiae

Returns 9.012 field

Return type str

Usage:

```
>>> from NIST.fingerprint.functions import lstTo137
>>> lstTo137(
...     [[  1,  7.85,  7.05, 290, 0, 100 ],
...      [  2, 13.80, 15.30, 155, 0, 100 ],
...      [  3, 11.46, 22.32, 224, 0, 100 ],
...      [  4, 22.61, 25.17, 194, 0, 100 ],
...      [  5,  6.97,  8.48, 153, 0, 100 ],
...      [  6, 12.58, 19.88, 346, 0, 100 ],
...      [  7, 19.69, 19.80, 111, 0, 100 ],
...      [  8, 12.31,  3.87, 147, 0, 100 ],
...      [  9, 13.88, 14.29, 330, 0, 100 ],
...      [ 10, 15.47, 22.49, 271, 0, 100 ]],
...     500
... )

↪'1\x1f154\x1f138\x1f290\x1f0\x1f100\x1e2\x1f271\x1f301\x1f155\x1f0\x1f100\x1e3\x1f225\x1f43
↪'
```

NIST.fingerprint.functions.**mm2px**(*data*, *res*)

Transformation the coordinates from millimeters to px.

Parameters

- **data** (*tuple*) – Coordinate value.
- **res** (*int*) – Resolution in DPI

Returns Transformed coordinates.

Return type list of float

Usage:

```
>>> from NIST.fingerprint.functions import mm2px
>>> mm2px( ( 12.7, 12.7 ), 500 )
[250.0, 250.0]
```

NIST.fingerprint.functions.**px2mm**(*data*, *res*)

Transformation the coordinates from pixel to millimeters

Parameters

- **data** (*tuple*) – Coordinate value.
- **res** (*int*) – Resolution in DPI

Returns Transformed coordinates.

Return type list of float

Usage:

```
>>> from NIST.fingerprint.functions import px2mm
>>> px2mm( ( 250, 250 ), 500 )
[12.7, 12.7]
```

NIST.fingerprint.functions.**tetraptych**(*mark*, *pr*, *markidc=-1*, *pridc=-1*)

Return an image with the mark and the print in the first row, and the corresponding images annotated in the second row.

Parameters

- **mark** (NIST (page 5)) – Latent NIST object.
- **pr** (NIST (page 5)) – Print NIST object.
- **markidc** (*int*) – IDC value for the latent NIST object.
- **pridc** (*int*) – IDC value for the print NIST object.

Usage:

```
>>> from NIST.fingerprint.functions import tetraptych
>>> img = tetraptych( sample_type_13, sample_type_4_tpcard, pridc = 1 )
>>> img# doctest: +ELLIPSIS
<PIL.Image.Image image mode=RGB size=1604x1536 at ...>
```

```
>>> from hashlib import md5
>>> md5( img.tobytes() ).hexdigest()
'3779a044e5a56b56380a895a9a5297ea'
```

1.5 Plugins manual

Marco De Donno's Fingerprint module overloading

This section contains all the docstring related to the NIST_MDD format, implementing vendor specific functions. This documentation is automaticaly generated, and used as unittest.

NIST_MDD module

class NIST.plugins.MDD.**NIST_MDD**(**args*, ***kwargs*)

Bases: *NIST.fingerprint.NISTf* (page 6)

Overload of the *NIST.fingerprint.NISTf()* (page 6) class, to implement all functions related to the pairing information, from the function to store the pairingn information in the NIST file (in the user-defined field 9.225), to the function to annotate the pairing on the images.

__module__ = 'NIST.plugins.MDD'

add_Type09(*minutiae=None*, *idc=0*, ***options*)

Overload of the *NIST.fingerprint.NISTf.add_Type09()* (page 6) function to initialize the AnnotationList with pairing information if provided.

See also:

NIST.fingerprint.NISTf.add_Type09() (page 6)

add_pairing(*lst*, *idc=-1*)

Add the pairing information into the AnnotationList passed in argument.

Parameters

- **lst** (AnnotationList (page 34)) – List of Annotations to process
- **idc** (*int*) – IDC value

Returns Updated AnnotationList

Return type *AnnotationList* (page 34)

annotate(*image*, *data*, *type='minutiae'*, *res=None*, *idc=-1*, ***options*)

Overloading of the NISTf.annotate() function to incorporate the annotation of the paired minutiae in yellow.

Parameters

- **image** (*PIL.Image*) – Image to annotate.
- **data** (AnnotationList (page 34)) – Data used to annotate the image.
- **type** (*str*) – Type of annotation (minutiae or center).
- **res** (*int*) – Resolution of the image in DPI.

Returns Annotated image.

Return type PIL.Image

checkMinutiae(*idc=-1*)

Overload of the *NIST.fingerprint.NISTf.checkMinutiae()* (page 7) function, to set the pairing information.

Parameters **idc** (*int*) – IDC value.

See also:

NIST.fingerprint.NISTf.checkMinutiae() (page 7)

get_latent_annotated(*idc=-1*, ***options*)

Overloading of the NISTf.get_latent_annotated() function to incorporate a special annotation for paired minutiae.

Parameters **idc** (*int*) – IDC value.

Returns Annotated latent fingermark

Return type PIL.Image

Usage:

```
>>> mark.get_latent_annotated() # doctest: +ELLIPSIS
<PIL.Image.Image image mode=RGB size=500x500 at ...>
```

get_minutiae (*format=None*, *idc=-1*, ***options*)
Overload of the *NIST.fingerprint.NISTf.get_minutiae()* (page 16) function to add the pairing number.

Parameters

- **format** (*str or list*) – Format to return the minutiae.
- **idc** (*int*) – IDC value.

Returns List of minutiae.

Return type *AnnotationList* (page 34)

See also:

NIST.fingerprint.NISTf.get_minutiae() (page 16)

get_minutiae_by_pairing_name (*name*, *format=None*, *idc=-1*)
Filter the minutiae list by pairing name.

Parameters

- **name** (*list*) – Name of the Annotations to retrieve.
- **format** (*str or list*) – Format of the Annotations to retrive.
- **idc** (*int*) – IDC value.

Returns Filtered AnnotationList

Return type *AnnotationList* (page 34)

Usage:

```
>>> mark.get_minutiae_by_pairing_name( [ 1, 2 ] ) # doctest: +NORMALIZE_
↪WHITESPACE
[
    Minutia( i='1', x='7.85', y='7.05', t='290', q='0', d='A', n='1' ),
    Minutia( i='2', x='13.8', y='15.3', t='155', q='0', d='A', n='2' )
]
```

get_minutiae_paired (*format=None*, *idc=-1*, ***options*)
Return all minutiae which are paired.

get_minutiae_paired_all (*format=None*, ***options*)
Return only the paired minutiae for all fingers, similair to the get_minutiae_all() function

get_minutiae_paired_count (*idc=-1*)
Get the number of minutiae pairing.

Parameters **idc** (*int*) – IDC value.

Returns Number of minutiae paired

Return type int

```
>>> mark.get_minutiae_paired_count()
3
```

get_pairing (*idc=-1*, *clean=False*)
Return the pairing information (minutia id, pairing id). This information is stored in the field 9.225.

Parameters

- **idc** (*int*) – IDC value.

- **clean** (*boolean*) – Remove the minutiae without pairing information ('None' value as pairing name otherwise).

Usage:

```
>>> mark.get_pairing() # doctest: +NORMALIZE_WHITESPACE
[
    Pairing( i='1', n='1' ),
    Pairing( i='2', n='2' ),
    Pairing( i='3', n='3' ),
    Pairing( i='4', n='None' ),
    Pairing( i='5', n='None' ),
    Pairing( i='6', n='None' ),
    Pairing( i='7', n='None' ),
    Pairing( i='8', n='None' ),
    Pairing( i='9', n='None' ),
    Pairing( i='10', n='None' )
]
```

```
>>> mark.get_pairing( clean = True ) # doctest: +NORMALIZE_WHITESPACE
[
    Pairing( i='1', n='1' ),
    Pairing( i='2', n='2' ),
    Pairing( i='3', n='3' )
]
```

get_print_annotated(*idc=-1*)

Overloading of the NISTf.get_print_annotated() function to incorporate a special annotation for paired minutiae.

Parameters **idc** (*int*) – IDC value.

Returns Annotated fingerprint

Return type PIL.Image

Usage:

```
>>> pr.get_print_annotated( 1 ) # doctest: +ELLIPSIS
<PIL.Image.Image image mode=RGB size=500x500 at ...>
```

migrate_Type13_to_type14(*idc=-1*)

Function to migrate an latent fingermark to a fingerprint (Type13 to Type14).

Parameters **idc** (*idc*) – IDC value.

set_minutiae(*data*, *idc=-1*)

Overload of the set_minutaie() function, to set the pairing information at the same time.

See also:

NIST.fingerprint.NISTf.set_minutiae() (page 28) NIST.fingerprint.NISTf.set_pairing()

set_pairing(*pairing=None*, *idc=-1*, ***options*)

Function to set the pairing information in the User-defined field 9.225. The pairing information is stored as following:

minutia id <US> minutia name <RS> ...

Parameters **pairing** (AnnotationList (page 34)) – Pairing information.

Let the pairing information be defined as follow:

```
>>> from NIST.fingerprint.functions import AnnotationList
>>> data = [
...     ( '1', '1' ), # Minutiae '1' nammed '1'
...     ( '2', '2' ), # Minutiae '2' nammed '2'
...     ( '3', '3' )  # Minutiae '3' nammed '3'
... ]
```

The pairing is set as follow:

```
>>> mark2 = mark.get()
>>> mark2.set_pairing( data )
```

The pairing can also be set with an AnnotationList object:

```
>>> pairing = AnnotationList()
>>> pairing.from_list( data, format = "in", type = "Pairing" )
>>> pairing # doctest: +NORMALIZE_WHITESPACE
[
    Pairing( i='1', n='1' ),
    Pairing( i='2', n='2' ),
    Pairing( i='3', n='3' )
]
```

The pairing is set as follow:

```
>>> mark2.set_pairing( pairing )
```

Morpho's Fingerprint module overloading

This section contains all the docstring related to the NIST_Morpho format, implementing vendor specific functions. This documentation is automaticaly generated, and used as unittest.

NIST_Morpho module

class NIST.plugins.Morpho.**NIST_Morpho**(**args*, ***kwargs*)

Bases: *NIST.fingerprint.NISTf* (page 6)

Overload of the *NIST.fingerprint.NISTf()* (page 6) class to implement functions specific to Morpho NIST files.

__module__ = 'NIST.plugins.Morpho'

export_cfv(*file*, *idc=-1*)

Export the CFV content to a file on disk.

Parameters

- **file** (*string*) – File to export to.
- **idc** (*int*) – IDC value.

get_caseName(*idc=-1*)

Return the case name (field 2.007).

Returns Case name.

Return type str

get_cfv(*idc=-1*)

Return the CFV files used for by the matcher.

Parameters **idc** (*int*) – IDC value.

Returns Binary CFV file.

Return type Bitstring

get_cores(*idc=-1*)

Process and return the coordinate of the cores.

Parameters **idc** (*int*) – IDC value.

Returns List of cores

Return type *AnnotationList* (page 34)

Usage:

```
>>> sample_type_9_10_14.get_cores() # doctest: +NORMALIZE_WHITESPACE
[
    Core( x='16.66', y='22.78' )
]
```

The function returns 'None' if no cores are stored in the NIST object.

```
>>> sample_type_4_tpcard.get_cores() == None
True
```

get_delta(*idc=-1*)

Process and return the coordinate of the deltas.

Parameters **idc** (*int*) – IDC value.

Returns List of deltas

Return type *AnnotationList* (page 34)

get_jar(*idc=-1*)

Get the content of all files present in the JAR file stored in the field 9.184. The returned dictionnary contains the as follow:

```
{
    'file name': 'file content',
    ...
}
```

The content of the files are not parsed, but returned as string value.

Parameters **idc** (*int*) – IDC value.

Returns Content of all files stored in the JAR file.

Return type dict

get_minutiae(*format=None*, *idc=-1*, ***options*)

Overload of the NISTf.get_minutiae() function to extract the information from the JAR stored in the 9.184 field. The minutiae coordinates are converted in the correct format (mm in this case), and stored in an AnnoationList object. All functions based on this function should work normaly.

is_binary(*ntype*, *tagid*)

Add some binary fields to the list of fields to format with `NIST.traditional.functions.bindump()`.

process_imageenh(*idc=-1*)

Function to process the imgageenh.2 content stored in the jar field. This function replays the actions done by the user. The result is the final annotation as seen in the MBIS interface.

Parameters **idc** (*int*) – IDC value.

Returns Dictionnary of the processed data.

Return type python dict

set_caseName(*name*, *idc=-1*)
Set the case name field (field 2.007).

Parameters **name** – Name of the case to set in the field 2.007

ULWLQMetric Fingerprint module overloading

This section contains all the docstring related to the ULWLQMetric format, implementing vendor specific functions. This documentation is automaticaly generated, and used as unittest.

ULWLQMetric module

class NIST.plugins.ULWLQMetric.**NISTULWLQMetric**
Bases: object

__dict__ = dict_proxy({'__dict__': <attribute '__dict__' of 'NISTULWLQMetric' objec

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

__module__ = 'NIST.plugins.ULWLQMetric'

__weakref__
list of weak references to the object (if defined)

1.6 Changelog

Pre-release

Develop

Bugfix:

- Change the field used for pairing from 9.255 to 9.225.

Add:

- Add the *NIST.fingerprint.functions.diptych()* (page 42) function.
- Add the functions to process the LQMetic data generated with ULW.
- Add the *tool* directory (AN-FieldDefinition parser (and other scripts in the future)).

Remove:

- Remove the deprecated functions in the NIST class.

Change:

- Split the main NIST object into the NIST_core and NIST_traditional parts.
- Move the MDD, Morpho and ULWLQMetric NIST classes to the plugin module.
- Misc refactoring.

Patch:

- Remove the duplicates minutiae in the NIST_Morpho class.

Commits since latest release:

- **Update the documentation for the get_tenprintcard_{front,back} functions** by *Marco De Donno* at *2021-06-27 16:48:09*
- **Add the doctest (and related nist file) for the get_palmar() function** by *Marco De Donno* at *2021-06-27 16:16:34*

- **Use the string_to_hex() function instead of testing a raw string in the changeFormatImage()** by *Marco De Donno* at *2021-06-27 14:00:11*
- **Add the parameter types for the has_idc() function** by *Marco De Donno* at *2021-06-27 13:55:42*
- **Update the documentation for the __str__() function** by *Marco De Donno* at *2021-06-27 13:53:01*
- **Add the ntypeNotFound exception check in the __str__() function** by *Marco De Donno* at *2021-06-27 13:52:35*
- **Documentation update for the get_minutiaeCount() function** by *Marco De Donno* at *2021-06-27 13:46:31*
- **Check the idc passed in parameter in the get_minutiaeCount() function** by *Marco De Donno* at *2021-06-27 13:46:11*
- **Update the documentation for the checkIDC() function** by *Marco De Donno* at *2021-06-27 13:45:37*
- **Update the documentation for the get_minutia_by_id() function** by *Marco De Donno* at *2021-06-27 13:01:21*
- **Add the documentation for the process_minutiae_field() function** by *Marco De Donno* at *2021-06-27 13:00:53*
- **Doc update for the get_fpc_list() function** by *Marco De Donno* at *2021-06-26 22:37:44*
- **Add support for multi-FPC in one field for the get_fpc_list() function** by *Marco De Donno* at *2021-06-26 22:37:00*
- **Add the documentation for the get_idc_for_fpc() function** by *Marco De Donno* at *2021-06-26 22:19:07*
- **Add the documentation for the split() function** by *Marco De Donno* at *2021-06-26 19:55:12*
- **Documentation update for te decode_fgp function** by *Marco De Donno* at *2021-06-26 19:45:15*
- **Check for values smaller than 0 in the encode_fgp function** by *Marco De Donno* at *2021-06-26 19:44:54*
- **Add the 'separator' argument for the encode_fgp function and refactoring** by *Marco De Donno* at *2021-06-26 19:44:10*
- **Update the documentation for the decode_fgp function** by *Marco De Donno* at *2021-06-26 19:43:00*
- **Update the encode_gca() documentation** by *Marco De Donno* at *2021-06-26 18:21:15*
- **Update the documentation for the printableFieldSeparator function** by *Marco De Donno* at *2021-06-26 18:15:06*
- **Only apply the string substitution to strings in the printableFieldSeparator function** by *Marco De Donno* at *2021-06-26 11:43:12*
- **Skip empty fields on load - Missing ending 1C** by *Marco De Donno* at *2021-06-25 21:34:54*

  This is done because of a US NIST file that was off by one byte at the end of the file. This can append if the last byte is not a "1C". If this skip produce a valid NIST object (i.e there is not fields after), it's probable that the entire NIST file has been processed correctly.

  If this error is present in the middle of the file, the processing of the next record should fail and crash completely the loading of the file (hopefully...).

- **Patch the decoding of fields containing ":" in the data** by *Marco De Donno* at *2021-06-25 20:15:43*

  The split is done only once on each field; this has the effect of splitting only with the first ":" and not with the ones that are present as data.

  The documentation is updated to reflect this new decoding of the files.

- **Show the full output for the dump() function in the doc** by *Marco De Donno* at *2021-06-25 19:13:58*

  This allows to check every fields in the load and dump process.

- **Add support for the JPEG2KC format** by *Marco De Donno* at *2021-06-25 14:19:17*

  This file format abbreviation was found in a US NIST files. The field contained a JPEG2000 file. I can only suppose that the format was LOSSY, but I can not confirm nor disprove it. The file will be assumed to be a JPEG2000 LOSSY.

- **Doc update** by *Marco De Donno* at *2021-06-17 18:54:56*
- **Check for non-existing ntypes in the get_idc() function** by *Marco De Donno* at *2021-06-17 18:54:27*
- **Set the idc as optional in the get_tagsid() function call** by *Marco De Donno* at *2021-06-17 18:15:48*
- **Add the doctest for the get_identifier() function** by *Marco De Donno* at *2021-06-17 16:58:10*
- **Add the documentation for the get_tagsid() function** by *Marco De Donno* at *2021-06-17 16:55:44*
- **Remove the doctest for the migrate_Type04_to_Type14() function** by *Marco De Donno* at *2021-06-10 11:20:35*
- **Patch the documentation for the migrate_Type04_to_Type14 function** by *Marco De Donno* at *2021-06-09 15:01:39*
- **Doc update** by *Marco De Donno* at *2021-06-08 22:03:51*
- **Merge branch 'feature/doctest_on_official_nist_files' into develop** by *Marco De Donno* at *2021-06-08 18:55:19*
- **Patch the migrate_Type04_to_Type14() function doctest** by *Marco De Donno* at *2021-06-08 18:45:34*
- **Merge branch 'develop' into feature/doctest_on_official_nist_files** by *Marco De Donno* at *2021-06-08 18:45:15*
- **Add support for the FGP multi field for the type4 to type14 migration process** by *Marco De Donno* at *2021-06-08 18:43:12*
- **Use the /tmp folder to test the export of the print and print_annotated functions** by *Marco De Donno* at *2021-06-08 18:40:54*
- **Doc update - use of the BioCTS sample files** by *Marco De Donno* at *2021-06-07 22:42:24*

  This commit uses ANSI/NIST files fro the NIST website. The files are public and available with the BioCTS for ANSI/NIST-ITL software.

- **Update the gitignore list to include the sample files** by *Marco De Donno* at *2021-06-07 22:37:20*
- **Force the convertion to RGB image when requesting an annotated image** by *Marco De Donno* at *2021-06-07 22:24:10*
- **Only manipulates minutiae in the crop process if some are present** by *Marco De Donno* at *2021-06-07 22:22:54*
- **Add the possible Ridge Count field in the minutiae tag** by *Marco De Donno* at *2021-06-07 22:15:20*

  Reading the NIST standard, the ridge count can be present in the minutiae field. By simply splitting the field with the normal separator, it's not possible, without counting the number of fields retrieved, to know if the RC field is present or not, hence not possible to use the variable unpacking.

- **Force the clean of the NIST object on load** by *Marco De Donno* at *2021-06-07 22:13:52*
- **Remove the critical warning for unknown Types** by *Marco De Donno* at *2021-06-07 20:38:06*
- **Remove the doctest2 files** by *Marco De Donno* at *2021-06-05 19:17:28*
- **Decode the multi-field fgp field for Type-04** by *Marco De Donno* at *2021-06-05 18:34:33*
- **Add the 'only_first' parameter to the NIST.core.functions.decode_fgp() function** by *Marco De Donno* at *2021-06-05 18:34:01*
- **Patch the get_tenprintcard_front function for Type04 NIST files** by *Marco De Donno* at *2021-06-05 15:18:50*

- **Merge branch 'fix/multi_finger_position_code' into develop** by *Marco De Donno* at *2021-06-05 14:14:59*
- **Add support for multiple Finger Position Code for Types3-6** by *Marco De Donno* at *2021-06-05 14:08:13*

  This commit implements the storage of multiple possible Finger Position Codes for the Types3-6, field 004. The field 004 is described as follow in the NIST documentation (version 2007):

  11.2.4 Finger position (FGP)

  This mandatory fixed-length field of six binary bytes shall occupy the seventh through twelfth byte positions of each record type. It shall contain possible finger positions beginning in the leftmost byte of the field (byte seven of the record). The decimal code number corresponding to the known or most probable finger position shall be taken from Table12 (only finger numbers 0-14 apply to Types 3-6) and entered as a binary number right justified and left zero filled within the eight-bit byte. Up to five additional finger positions may be referenced by entering the alternate finger positions in the remaining five bytes using the same format. If fewer than five finger position references are to be used, the unused bytes shall be filled with the binary equivalent of "255". The code "0", for "Unknown Finger", shall be used to reference every finger position from one through ten.
- **Add the test for the NIST.core.functions module** by *Marco De Donno* at *2021-06-05 14:02:30*
- **Split the debugging information for the GCA decoding** by *Marco De Donno* at *2021-06-04 18:11:26*

  This will help the debugging process for non-valid GCA values.
- **Add the fingerpint.get_fpc_list() function** by *Marco De Donno* at *2021-05-19 18:26:54*

  This function get the list of all fpc present in the NIST file, regardless of the idc related to the fpc. This is useful to iterate on all fpc available.
- **Add the SEGMENTS_POSITION_CODE variables in the labels** by *Marco De Donno* at *2021-02-28 23:33:46*
- **Patch the convertion to type 14 - wrong field for the cga** by *Marco De Donno* at *2020-05-24 17:58:41*
- **Update the requirements file (add numpy and sort the list)** by *Marco De Donno* at *2020-05-22 17:51:59*
- **Remove useless fpc parameter in the get_minutiae_paired_all function** by *Marco De Donno* at *2020-05-16 17:55:09*
- **Merge branch 'develop' of https://github.com/mdedonno1337/NIST into develop** by *Marco De Donno* at *2020-05-16 17:45:32*
- **Add the get_minutiae_paired_all function to the MDD module** by *Marco De Donno* at *2020-05-16 17:43:51*
- **Merge branch 'develop' of esc-md-git.unil.ch:/mdedonno/NIST into develop** by *Marco De Donno* at *2020-04-30 14:45:53*
- **Remove the link to readthedocs** by *Marco De Donno* at *2020-04-30 14:45:12*
- **Typo** by *Marco De Donno* at *2020-04-29 23:07:47*
- **Update the installation process** by *Marco De Donno* at *2020-04-25 19:36:26*
- **Add the finger position code and palm position code label list** by *Marco De Donno* at *2019-12-19 16:47:03*
- **Add the generic front tenprint card** by *Marco De Donno* at *2019-11-21 19:51:06*
- **Typo** by *Marco De Donno* at *2019-11-19 21:17:42*
- **Patch: patch the is_initialized() function** by *Marco De Donno* at *2019-09-03 22:01:24*
- **Add the is_initialized() function** by *Marco De Donno* at *2019-09-03 21:51:55*
- **Add support for buffer and file in the loading function** by *Marco De Donno* at *2019-08-20 19:14:31*
- **Remove useless files** by *Marco De Donno* at *2019-08-08 17:18:49*

- **Merge branch 'develop' of esc-md-git.unil.ch:mdedonno/NIST into develop** by *Marco De Donno* at *2019-08-05 15:31:41*
- **Add support for buffer loading** by *Marco De Donno* at *2019-08-05 15:27:12*
- **Remove the fuckit library from the dependencies** by *Marco De Donno* at *2019-08-03 21:04:28*
- **Change the default set_print image format to RAW (instead of the WSQ one), and docupdate** by *Marco De Donno* at *2019-06-24 14:04:28*
- **Change the default compression algorithm used for type 4 and 13** by *Marco De Donno* at *2019-06-23 19:13:35*
- **Change the default return format of the image by the get_latent function** by *Marco De Donno* at *2019-06-22 11:39:06*
- **Pass the arguments of the get_image to the latent, print and palmar functions** by *Marco De Donno* at *2019-06-22 11:38:48*
- **Refactoring and add of the palmar image support in the NISTf.get_image() function** by *Marco De Donno* at *2019-06-17 17:46:04*
- **Change the signature of the get_image function** by *Marco De Donno* at *2019-06-17 16:41:44*

  The kwargs value is passed to the get_latent and get_print functions
- **Change the default level of logging** by *Marco De Donno* at *2019-06-13 01:38:32*
- **Dont allow the TGA format in the auto-decoding of the str images** by *Marco De Donno* at *2019-06-12 16:36:53*

v0.23.1 (2017-04-06)

Major bugfix:

- Revert the commit 728a91bfa26101f229a0c60bacb9723cfe3d2f91

v0.23 (2017-04-04)

Add:

- Add support for loading dict objects in NIST.
- Add support for tuple in the `NIST.MDD.NIST_MDD.set_pairing()` function.
- Add the *`NIST.fingerprint.NISTf.get_latent_hull()`* (page 15) function.
- Add the *`NIST.fingerprint.functions.Point()`* (page 41) and *`NIST.fingerprint.functions.Delta()`* (page 41) objects.
- Overload the '>>' operator for the *`NIST.fingerprint.functions.Annotation()`* (page 30) object.
- Add the function to sort `NIST.fingerprint.functions.AnnotationsList()` object based on the distance to a particular point.
- Add the `NIST.MDD.NIST_MDD.unpair()` function.
- Add the *`NIST.fingerprint.NISTf.get_delta()`* (page 12) function, and the annotation of the deltas in the *`NIST.fingerprint.NISTf.annotate()`* (page 6) function.
- Add the `NIST.Morpho.NIST_Morpho.process_imageenh()` and `NIST.Morpho.NIST_Morpho.get_delta()` functions.
- Add support for WSQ as input for the *`NIST.fingerprint.functions.changeFormatImage()`* (page 41) function.

- Add the `NIST.ULWLQMetric.NISTULWLQMetric.get_minutiae_by_LQM()`, `NIST.ULWLQMetric.NISTULWLQMetric.get_LQMetric_map()` functions.
- Add the LQMetric Quality Map if available.
- Add support for WSQ in the Fingerprint module.
- Add support for bifurcations and undetermined minutiae type.
- Add support for the Core annotations in the Morho_NIST() class.
- Add the *NIST.fingerprint.NISTf.export_latent_diptych()* (page 10) function.

Change:

- Overload of the set_minutiae in the NIST_MDD class (to set the pairing information).
- The *NIST.fingerprint.NISTf.get_latent_triptych()* (page 15) function returns the convex hull by default.
- Set the resolution in the latent and print exported images.
- Move the processing of the field 9.331 into the NISTf module.
- Use the builtin super() function instead of the future.super() function.
- Add **options to the `NIST.MDD.NIST_MDD.get_minutiae()` function.
- Remove the change of the NIST standard version (field 1.003) in the cleaning function.
- Move the processing of the 1.011 and 1.012 filds in the Fingerprint module.
- Made the WSQ module optional (raise an exception if not loaded AND used).

v0.22 (2017-01-26)

Add:

- Add `NIST.traditional.NIST.to_dict()` function.
- Add `NIST.traditional.NIST.get_all_tagid()` function.
- Add `NIST.traditional.NIST.to_json()` and `NIST.traditional.NIST.from_json()` functions.
- Add *NIST.fingerprint.functions.AnnotationList.remove()* (page 39) function.
- Add *NIST.fingerprint.functions.changeFormatImage()* (page 41) function.
- Add *NIST.fingerprint.NISTf.get_tenprint_annotated()* (page 21) function.
- Add *NIST.fingerprint.NISTf.get_image_annotated()* (page 14) function.
- Add auto funcitons (*NIST.fingerprint.NISTf.export_auto()* (page 9), *NIST.fingerprint.NISTf.export_auto_annotated()* (page 9) and *NIST.fingerprint.NISTf.crop_auto()* (page 9)).
- Add `NIST.MDD.NIST_MDD.get_minutiae_paired_count()` function.
- Add `NIST.Morpho.NIST_Morpho.get_jar()` and `NIST.Morpho.NIST_Morpho.get_cfv()` functions.
- Add `NIST.Morpho.NIST_Morpho.get_minutiae()` to process the 9.184 field.
- Add support for unicode data in the *load_auto()* (page 5) function.
- Add support for Type-13 NIST object as reference in the *NIST.fingerprint.functions.tetraptych()* (page 44) function.

Remove:

- Remove the `NIST.fingerprint.NISTf.has_field()` function.

Change:

- The input data of the *lstTo012()* (page 42) function have to be an AnnotationList or list object.
- The *checkMinutiae()* (page 7) returns the checks for all IDC if multiple IDC are stored in the NIST object.
- The `__str__()` function do not raise an exception if the NIST object is not initialized ("NIST object not initialized..." string instead).
- Recode of the `NIST.traditional.functions.tagSplitter()` function: The common tests are done inside the function, instead of outside.
- Recode of the *NIST.fingerprint.NISTf.set_cores()* (page 26) function.
- Recode of the `NIST.MDD.NIST_MDD.set_pairing()` function.

Patch:

- *NIST.fingerprint.NISTf.get_minutia_by_id()* (page 15): Error if the *format* parameter is passed to *None*.
- *NIST.traditional.NIST.load()* (page 5) function: If the tagid is 999, the tag was not updated correctly (error possible if the length of the tag differs).
- The width and the height variables were inverted int he *NIST.fingerprint.NISTf.get_tenprint()* (page 20) function.
- Critical bug in the pairing information update `NIST.MDD.NIST_MDD.set_pairing()` function.

v0.21 (2016-12-25)

Add:

- First Sphinx documentation.
- Add the `NIST.traditional.NIST.get()` function.
- Add support for keyword arguments for the *Annotation* object.
- Add the parameter in the *NIST.fingerprint.functions.AnnotationList.from_list()* (page 37) class to select the type of *Annotation* stored (*Annotation*, *Minutia* or *Core*).
- Add *NIST.fingerprint.NISTf.migrate_Type04_to_Type14()* (page 25) function.
- Add `NIST.MDD.NIST_MDD.migrate_Type13_to_type14()` function.
- Add `NIST.traditional.NIST.get_field_multi_idc()` and `NIST.traditional.NIST.get_fields_multi_idc()` functions.
- Add *NIST.fingerprint.NISTf.get_tenprint()* (page 20) function.
- Add `NIST.fingerprint.NISTf.get_tenprintcard()` function (European format).

Change:

- Migrate the `NIST.MDD.NIST_MDD.get_pairing()` function to return *AnnotationList()* (page 34).
- Change the location to store the default annotation format (NIST.fingerprint.functions.Annotation.defaultformat).

Patch:

- `NIST.traditional.functions.bindump()` was returning the first byte at the end of the string

v0.20 (2016-12-18)

Add:

- Add the *init_latent()* (page 21), *init_print()* (page 24) and *init_new()* (page 22) functions in the NISTf() class.
- Add options in various functions to allow func(params) calls.
- Add the `NIST.traditional.NIST.merge()` function.
- Add support for the Type14 in various functions.

Remove:

- Remove deprecated functions.

Change:

- The `NIST.MDD.NIST_MDD.add_Type09()` overload his super() function to set the pairing information.
- Change default standard value to '0300'.
- Store the result of the self.get_ntype() if multiple calls have to be done in the same function (performance issues).

v0.19 (2016-12-01)

Add:

- Add the *Annotation()* (page 30), *Minutia()* (page 41) and *Core()* (page 40) classes.
- Add the *AnnotationList()* (page 34) class.

Change:

- Change the format of the Minutiae and Core stored in the NIST object (from list to Minutia() and Core() object).

v0.18 (2016-11-22)

Add:

- Add the `add_type14()` function with default values.
- Add the NIST_MDD class to process NIST objects with pairing information (user-defined field 9.255), and add function to annotate the pairing information (`NIST.MDD.NIST_MDD.annotate()`, `NIST.MDD.NIST_MDD.get_latent_annotated()` and `NIST.MDD.NIST_MDD.get_print_annotated()`).
- Add support for PNG format.

Change:

- Split the *NIST.fingerprint.NISTf.crop()* (page 8) function in *crop_latent()* (page 9) and *crop_print()* (page 9) functions.
- Delete all empty IDC and ntypes in the clean() function.
- Optimization of the *get_latent_annotated()* (page 14), *get_print_annotated()* (page 20) and `get_annotated()` functions.

Delete:

- Remove the minutiae_filter() function.

Patch:

- Monkey-patch of the x.001 field (not in agreement with the NIST standard).

v0.17 (2016-09-20)

Add:

- Add the *NIST.fingerprint.NISTf.set_cores()* (page 26) function.
- Add support for Type04 in the *NIST.fingerprint.NISTf.set_resolution()* (page 29) function.
- Add the `NIST.traditional.NIST.update()` function.

Change:

- Auto-increment of the Minutia id if not provided.
- Annotate all the cores if multiple cores are stored in the NIST object.
- Automatic selection of the correct diptych with the *NIST.fingerprint.NISTf.get_diptych()* (page 12) function.
- Change the *NIST.traditional.NIST.data* variable to *MDmisc.RecursiveDefaultDict.defDict()*.
- Clean the NIST object before printing.
- Check the IDC field (x.002) for each ntype in the clean() function.
- Patch the fields 1.011 and 1.012 according to the NIST standard.

v0.16 (2016-08-17)

Add:

- Add the *NIST.fingerprint.NISTf.crop_latent()* (page 9).
- Add support for uncompressed (RAW) Type-04 images, and adapt the 'changeResolution()' function.
- Add the function *NIST.fingerprint.NISTf.get_image()* (page 13), with automatic determination of the main type (latent or print image).
- Add the *NIST.fingerprint.NISTf.export_latent()* (page 9) and *NIST.fingerprint.NISTf.export_print()* (page 10), and the annotated equivalent functions.
- Add support to crop Type13 and Type04 images.

v0.15 (2016-08-14)

Add:

- Add the first attempt for NIST object with M1 annotation format.
- Add misc functions.
- Add the identifier field for the NIST object.
- Add some information in the head of the string-representation of the NIST object.
- Add support for the JPEG2K format for latent images.
- Add the formatter function to print formatted binary fields.

Change:

- Change the name of the main class in the Morpho module.
- Use of the Python3 division module.

Patch:

- Bug-fix in some docstrings.

v0.14 (2016-07-11)

Add:

- Add functions related to the image processing (get the latent annotated, quality map, . . .).
- Add functions to set values for latent and print (set_width(), set_height(), . . .).
- Add function to change the main class of the NIST object.
- Add support for passing tuple in the set_field() function instead of a tag-string.
- Add the functions get_diptych() and get_tetraptych()

Change:

- Change the format of the NIST object representation (non prittable character, . . .).
- Move deprecated functions.

v0.13 (2016-07-06)

Add:

- Add the function get_minutiae_all() to get the minutiae from all fingers in one list of lists.
- Add the function to add the case name (field 2.007) for Morpho NIST objects.

Change:

- Extend the NIST-fingerprint library to work with the ULWLQMetric module (added in the default installer).
- Move deprecated methods in the NISTf_deprecated. Need to import this class instead of NISTf if you want to use deprecated functions with the old code (not recommended !).
- Move the version of the standard in a class attribute (changeable on the fly).

Patch:

- Bug around : if the 1.003 field is malformed (an IDC is missing), the missing value is replaced with a 1.
- Misc patches.

v0.12 (2016-06-29)

Add:

- First version of the function to work with fingerprints (get_print() to get the PIL or WSQ image) from 4.999 WSQ encoded image.
- Update the names of the imports.
- Patch the get_minutiae() function (order of values).
- Deprecation of the get_image() function.

v0.11 (2016-06-26)

Add:

- Add the versioneer module to manage the version name in the setup file produced.

Change:

- Modification of the structure of the module: separation of the traditional module from the fingerprint functions.
- Change the parameter 'minutiae' in NIST.fingerprint.NISTf.add_Type09() to optional.
- Misc optimizations.

v0.10 (2016-06-25)

Add:

- Add function returning the number of minutiae present in the NIST object (field 9.010).
- Add function to add default empty records (Type-01, Type-02, Type-09 and Type-13).

Change:

- Update from distutils.core to setuptools
- Checker for the minutiae field (all minutiae have to be on the image).
- Code cleaning.

v0.9 (2016-06-19)

Add:

- Type-14, Type-10, Type-15, Type-16, Type-17, Type-18, Type-19, Type-20, Type-21, Type-98 and Type-99 parser

Change:

- Change in the structure of the package, and addition of the __init__ file to load the NIST package
- Rewriting of the doctest.py module.
- get_fields() and set_fields() to get and set multiple fields at once
- Documentation of all fields (name and abbreviation)

v0.8 (2016-06-18)

Add:

- Function to delete a ntype, idc or field
- Type-04 dumper

Change:

- Updater for the File Content field (1.003) based on the content of the NIST object
- Field 4.005 checker (conformation to the standard)

Patch:

- Patch the type for the Type-04 data
- Bug fix for the unsupported Type (allowing to skip the record)

- Bug fix for the data in 999 tagid
- Bug fix for the order of the excepted ntype

v0.7 (2016-06-17)

Add:

- Type-04 parser
- get_caseName, get_center
- 'Grayscale Compression Algorithm' field decoder
- Get and set the latent friction ridge image (Type-13) in PIL and RAW format
- Functions for image format transform (PIL <-> RAW)

Change:

- Warning for unsupported Type

v0.6 (2016-06-17)

Add:

- Parser between the 9.012 field and the minutiae table
- Function to return the minutiae table

v0.5 (2016-06-16)

Add:

- Autoloader (to detect the type (NIST object or URI to file) and load it in memory
- @deprecated decorator for sort deprecation (backward compatibility)
- Function to check and return the IDC (if unique) of a ntype
- Function to get the content of a field via the field-tag
- tagSplitter function

v0.4 (2016-06-16)

Add:

- __str__ and __repr__ functions
- Function to write the NIST object to disk
- Function do dump the content of the NIST object in binary (ready to write to disk)
- Function to check some errors possible in the content of the NIST object
- Function to return the tag from a ntype and a tagid
- Function to clean the self.data (deletion of empty fields)
- Add some fields in the LABEL variable

Change:

- Set an IDC for Type-01, even if the standard says nothing about it. For simplification of the code later

v0.3 (2016-06-12)

Add:

- Function to dump the content of the NIST object in a string variable
- Add the leveler function
- Function to get the abbreviation and description of fields
- Functions to get the list of ntype and idc's

Change:

- Move the Logger class to the misc library.

v0.2 (2016-06-12)

Add:

- Function to process the File Content (field 1.003)
- LABEL of some fields
- Debug information
- Binary onscreen dumper

v0.1 (2016-06-11)

- First version of the parser for Type-01, Type-02, Type-09 and Type-13
- Initial commit

1.7 Contact

Marco De Donno
School of Criminal Justice
Faculty of Law, Criminal Justice and Public Administration
University of Lausanne, Switzerland
http://www.unil.ch/esc

Code author: Marco De Donno <Marco.DeDonno@unil.ch> <mdedonno1337@gmail.com>

References

[NIST2013] Data Format for the Interchange of Fingerprint, Facial & Other Biometric Information, NIST Special Publication 500-290 Rev1 (2013), https://www.nist.gov/itl/iad/image-group/ansinist-itl-standard-references

Python Module Index

n

NIST.fingerprint, 6
NIST.fingerprint.functions, 30
NIST.plugins.MDD, 44
NIST.plugins.Morpho, 48
NIST.plugins.ULWLQMetric, 50
NIST.traditional, 4

Index

Symbols

__add__() (*NIST.fingerprint.functions.Minutia method*), 41
__dict__ (*NIST.plugins.ULWLQMetric.NISTULWLQMetric attribute*), 50
__eq__() (*NIST.fingerprint.functions.Annotation method*), 30
__getattr__() (*NIST.fingerprint.functions.Annotation method*), 31
__getitem__() (*NIST.fingerprint.functions.Annotation method*), 31
__getitem__() (*NIST.fingerprint.functions.AnnotationList method*), 34
__iadd__() (*NIST.fingerprint.functions.Annotation method*), 32
__iadd__() (*NIST.fingerprint.functions.AnnotationList method*), 34
__init__() (*NIST.fingerprint.NISTf method*), 6
__init__() (*NIST.fingerprint.functions.Annotation method*), 32
__init__() (*NIST.fingerprint.functions.AnnotationList method*), 34
__init__() (*NIST.plugins.ULWLQMetric.NISTULWLQMetric method*), 50
__iter__() (*NIST.fingerprint.functions.Annotation method*), 32
__iter__() (*NIST.fingerprint.functions.AnnotationList method*), 35
__len__() (*NIST.fingerprint.functions.Annotation method*), 32
__len__() (*NIST.fingerprint.functions.AnnotationList method*), 35
__module__ (*NIST.fingerprint.NISTf attribute*), 6
__module__ (*NIST.fingerprint.functions.Annotation attribute*), 33
__module__ (*NIST.fingerprint.functions.AnnotationList attribute*), 35
__module__ (*NIST.fingerprint.functions.Core attribute*), 40
__module__ (*NIST.fingerprint.functions.Delta attribute*), 41
__module__ (*NIST.fingerprint.functions.Minutia attribute*), 41
__module__ (*NIST.fingerprint.functions.Point attribute*), 41
__module__ (*NIST.fingerprint.functions.dMinutia attribute*), 42
__module__ (*NIST.plugins.MDD.NIST_MDD attribute*), 45
__module__ (*NIST.plugins.Morpho.NIST_Morpho attribute*), 48
__module__ (*NIST.plugins.ULWLQMetric.NISTULWLQMetric attribute*), 50
__module__ (*NIST.traditional.NIST attribute*), 5
__repr__() (*NIST.fingerprint.functions.Annotation method*), 33
__repr__() (*NIST.fingerprint.functions.AnnotationList method*), 35
__rshift__() (*NIST.fingerprint.functions.Annotation method*), 33
__setattr__() (*NIST.fingerprint.functions.Annotation method*), 33
__setitem__() (*NIST.fingerprint.functions.AnnotationList method*), 35
__str__() (*NIST.fingerprint.functions.Annotation method*), 33
__str__() (*NIST.fingerprint.functions.AnnotationList method*), 35
__sub__() (*NIST.fingerprint.functions.Minutia method*), 41
__weakref__ (*NIST.plugins.ULWLQMetric.NISTULWLQMetric attribute*), 50

A

add_pairing() (*NIST.plugins.MDD.NIST_MDD method*), 45
add_Type04() (*NIST.fingerprint.NISTf method*), 6
add_Type09() (*NIST.fingerprint.NISTf method*), 6
add_Type09() (*NIST.plugins.MDD.NIST_MDD method*), 45
add_Type13() (*NIST.fingerprint.NISTf method*), 6
add_Type14() (*NIST.fingerprint.NISTf method*), 6
add_Type15() (*NIST.fingerprint.NISTf method*), 6
annotate() (*NIST.fingerprint.NISTf method*), 6
annotate() (*NIST.plugins.MDD.NIST_MDD method*), 45
Annotation (*class in NIST.fingerprint.functions*), 30
AnnotationList (*class in NIST.fingerprint.functions*), 34
append() (*NIST.fingerprint.functions.AnnotationList method*), 36
as_json() (*NIST.fingerprint.functions.Annotation method*), 33
as_json() (*NIST.fingerprint.functions.AnnotationList method*), 36
as_list() (*NIST.fingerprint.functions.Annotation method*), 34
as_list() (*NIST.fingerprint.functions.AnnotationList method*), 37

C

changeFormatImage() (*in module NIST.fingerprint.functions*), 41
changeResolution() (*NIST.fingerprint.NISTf method*), 7
checkMinutiae() (*NIST.fingerprint.NISTf method*), 7
checkMinutiae() (*NIST.plugins.MDD.NIST_MDD method*), 45

clean() (*NIST.fingerprint.NISTf method*), 8
clean() (*NIST.traditional.NIST method*), 5
Core (*class in NIST.fingerprint.functions*), 40
crop() (*NIST.fingerprint.NISTf method*), 8
crop_auto() (*NIST.fingerprint.NISTf method*), 9
crop_latent() (*NIST.fingerprint.NISTf method*), 9
crop_print() (*NIST.fingerprint.NISTf method*), 9

D

default_values() (*NIST.fingerprint.functions.dMinutia method*), 42
default_values() (*NIST.fingerprint.functions.Minutia method*), 41
defaultformat (*NIST.fingerprint.functions.Annotation attribute*), 34
defaultformat (*NIST.fingerprint.functions.Core attribute*), 41
defaultformat (*NIST.fingerprint.functions.Delta attribute*), 41
defaultformat (*NIST.fingerprint.functions.dMinutia attribute*), 42
defaultformat (*NIST.fingerprint.functions.Minutia attribute*), 41
defaultformat (*NIST.fingerprint.functions.Point attribute*), 41
Delta (*class in NIST.fingerprint.functions*), 41
diptych() (*in module NIST.fingerprint.functions*), 42
dMinutia (*class in NIST.fingerprint.functions*), 42
dumpbin() (*NIST.traditional.NIST method*), 5

E

export_auto() (*NIST.fingerprint.NISTf method*), 9
export_auto_annotated() (*NIST.fingerprint.NISTf method*), 9
export_cfv() (*NIST.plugins.Morpho.NIST_Morpho method*), 48
export_latent() (*NIST.fingerprint.NISTf method*), 9
export_latent_annotated() (*NIST.fingerprint.NISTf method*), 9
export_latent_diptych() (*NIST.fingerprint.NISTf method*), 10
export_print() (*NIST.fingerprint.NISTf method*), 10
export_print_annotated() (*NIST.fingerprint.NISTf method*), 10
export_print_diptych() (*NIST.fingerprint.NISTf method*), 10

F

filter_minutiae() (*NIST.fingerprint.NISTf method*), 10
from_list() (*NIST.fingerprint.functions.AnnotationList method*), 37

G

get() (*NIST.fingerprint.functions.Annotation method*), 34
get() (*NIST.fingerprint.functions.AnnotationList method*), 38
get_by_type() (*NIST.fingerprint.functions.AnnotationList method*), 38
get_caseName() (*NIST.plugins.Morpho.NIST_Morpho method*), 48
get_cfv() (*NIST.plugins.Morpho.NIST_Morpho method*), 48
get_compression() (*NIST.fingerprint.NISTf method*), 11
get_cores() (*NIST.fingerprint.NISTf method*), 12
get_cores() (*NIST.plugins.Morpho.NIST_Morpho method*), 49
get_delta() (*NIST.fingerprint.NISTf method*), 12
get_delta() (*NIST.plugins.Morpho.NIST_Morpho method*), 49
get_diptych() (*NIST.fingerprint.NISTf method*), 12
get_format() (*NIST.fingerprint.functions.AnnotationList method*), 39
get_fpc_list() (*NIST.fingerprint.NISTf method*), 12
get_height() (*NIST.fingerprint.NISTf method*), 12
get_idc_for_fpc() (*NIST.fingerprint.NISTf method*), 13
get_image() (*NIST.fingerprint.NISTf method*), 13
get_image_annotated() (*NIST.fingerprint.NISTf method*), 14
get_jar() (*NIST.plugins.Morpho.NIST_Morpho method*), 49
get_latent() (*NIST.fingerprint.NISTf method*), 14
get_latent_annotated() (*NIST.fingerprint.NISTf method*), 14
get_latent_annotated() (*NIST.plugins.MDD.NIST_MDD method*), 45
get_latent_diptych() (*NIST.fingerprint.NISTf method*), 15
get_latent_hull() (*NIST.fingerprint.NISTf method*), 15
get_latent_triptych() (*NIST.fingerprint.NISTf method*), 15
get_minutia_by_id() (*NIST.fingerprint.NISTf method*), 15
get_minutiae() (*NIST.fingerprint.NISTf method*), 16
get_minutiae() (*NIST.plugins.MDD.NIST_MDD method*), 46
get_minutiae() (*NIST.plugins.Morpho.NIST_Morpho method*), 49
get_minutiae_all() (*NIST.fingerprint.NISTf method*), 18
get_minutiae_by_pairing_name() (*NIST.plugins.MDD.NIST_MDD method*), 46
get_minutiae_by_type() (*NIST.fingerprint.NISTf method*), 18

get_minutiae_paired() (*NIST.plugins.MDD.NIST_MDD method*), 46
get_minutiae_paired_all() (*NIST.plugins.MDD.NIST_MDD method*), 46
get_minutiae_paired_count() (*NIST.plugins.MDD.NIST_MDD method*), 46
get_minutiaeCount() (*NIST.fingerprint.NISTf method*), 18
get_n_closest_from_point() (*NIST.fingerprint.functions.AnnotationList method*), 39
get_n_furthest_from_point() (*NIST.fingerprint.functions.AnnotationList method*), 39
get_pairing() (*NIST.plugins.MDD.NIST_MDD method*), 46
get_palmar() (*NIST.fingerprint.NISTf method*), 19
get_print() (*NIST.fingerprint.NISTf method*), 19
get_print_annotated() (*NIST.fingerprint.NISTf method*), 20
get_print_annotated() (*NIST.plugins.MDD.NIST_MDD method*), 47
get_print_diptych() (*NIST.fingerprint.NISTf method*), 20
get_resolution() (*NIST.fingerprint.NISTf method*), 20
get_size() (*NIST.fingerprint.NISTf method*), 20
get_tenprint() (*NIST.fingerprint.NISTf method*), 20
get_tenprint_annotated() (*NIST.fingerprint.NISTf method*), 21
get_tenprintcard_back() (*NIST.fingerprint.NISTf method*), 21
get_tenprintcard_front() (*NIST.fingerprint.NISTf method*), 21
get_width() (*NIST.fingerprint.NISTf method*), 21

H

hash() (*NIST.traditional.NIST method*), 5

I

init_latent() (*NIST.fingerprint.NISTf method*), 21
init_new() (*NIST.fingerprint.NISTf method*), 22
init_print() (*NIST.fingerprint.NISTf method*), 24
is_binary() (*NIST.plugins.Morpho.NIST_Morpho method*), 49

L

load() (*NIST.traditional.NIST method*), 5
load_auto() (*NIST.traditional.NIST method*), 5
lstTo012() (*in module NIST.fingerprint.functions*), 42
lstTo137() (*in module NIST.fingerprint.functions*), 43

M

migrate_Type04_to_Type14() (*NIST.fingerprint.NISTf method*), 25
migrate_Type13_to_type14() (*NIST.plugins.MDD.NIST_MDD method*), 47
Minutia (*class in NIST.fingerprint.functions*), 41
mm2px() (*in module NIST.fingerprint.functions*), 43
mm2px() (*NIST.fingerprint.NISTf method*), 25

N

NIST (*class in NIST.traditional*), 5
NIST.fingerprint (*module*), 6
NIST.fingerprint.functions (*module*), 30
NIST.plugins.MDD (*module*), 44
NIST.plugins.Morpho (*module*), 48
NIST.plugins.ULWLQMetric (*module*), 50
NIST.traditional (*module*), 4
NIST_MDD (*class in NIST.plugins.MDD*), 45
NIST_Morpho (*class in NIST.plugins.Morpho*), 48
NISTf (*class in NIST.fingerprint*), 6
NISTULWLQMetric (*class in NIST.plugins.ULWLQMetric*), 50

P

patch_to_standard() (*NIST.fingerprint.NISTf method*), 25
Point (*class in NIST.fingerprint.functions*), 41
process_imageenh() (*NIST.plugins.Morpho.NIST_Morpho method*), 49
process_minutiae_field() (*NIST.fingerprint.NISTf method*), 25
px2mm() (*in module NIST.fingerprint.functions*), 44
px2mm() (*NIST.fingerprint.NISTf method*), 26

R

remove() (*NIST.fingerprint.functions.AnnotationList method*), 39
reset_alpha_length() (*NIST.traditional.NIST method*), 5
reset_binary_length() (*NIST.traditional.NIST method*), 5

S

set_caseName() (*NIST.plugins.Morpho.NIST_Morpho method*), 49
set_cores() (*NIST.fingerprint.NISTf method*), 26
set_format() (*NIST.fingerprint.functions.Annotation method*), 34
set_format() (*NIST.fingerprint.functions.AnnotationList method*), 39
set_height() (*NIST.fingerprint.NISTf method*), 27
set_latent() (*NIST.fingerprint.NISTf method*), 27
set_latent_size() (*NIST.fingerprint.NISTf method*), 28
set_minutiae() (*NIST.fingerprint.NISTf method*), 28
set_minutiae() (*NIST.plugins.MDD.NIST_MDD method*), 47

set_pairing() (*NIST.plugins.MDD.NIST_MDD method*), 47
set_print() (*NIST.fingerprint.NISTf method*), 28
set_print_size() (*NIST.fingerprint.NISTf method*), 29
set_resolution() (*NIST.fingerprint.NISTf method*), 29
set_size() (*NIST.fingerprint.NISTf method*), 29
set_width() (*NIST.fingerprint.NISTf method*), 29
sort_dist_point() (*NIST.fingerprint.functions.AnnotationList method*), 40

T

tetraptych() (*in module NIST.fingerprint.functions*), 44

W

write() (*NIST.traditional.NIST method*), 5